

Андрей Гарнаев

Microsoft® Excel 2002 Разработка приложений

- Работа с базами данных средствами ADO
- Интеграция с офисными приложениями
- Программирование помощника MS Office
- Ответы на типовые вопросы разработчиков проектов
- Около 400 примеров

МАСТЕР_ПРОГРАММ_

Андрей Гарнаев

Microsoft® Excel 2002
Разработка приложений

Санкт-Петербург
«БХВ-Петербург»
2004

УДК 681.3.06
ББК 32.973.26-018.2
Г20

Гарнаев А. Ю.

Г20 MS Excel 2002: разработка приложений. — СПб.: БХВ-Петербург, 2003. — 768 с.: ил.

ISBN 5-94157-243-3

Книга предназначена для самостоятельного освоения средств, методов и нюансов создания профессиональных офисных проектов на основе MS Excel. Досконально разобрана объектная модель Excel, описан объектно-ориентированный подход к программированию офисных приложений, рассмотрены вопросы создания пользовательского интерфейса и справочной системы, автоматизации операций с рабочим листом и диаграммами, в том числе при обработке и анализе данных и принятии решений. Подробно изложены средства работы с файлами, методы интеграции офисных приложений, работа с Internet, применение XML и использование Excel в клиент-серверных приложениях. Показаны возможности Excel при работе с базами данных средствами ADO и в проектах Visual Basic .NET. Книга содержит около 400 примеров тщательно разработанных приложений: от создания пользовательских функций до построения информационных систем по сбору и обработке данных. Программный код в любом разделе самодостаточен и может быть непосредственно использован читателем при разработке собственных проектов. Книга написана на базе лекций, читаемых автором в Санкт-Петербургском государственном университете, а также на основе многолетнего личного опыта автора по разработке офисных проектов.

Для программистов, преподавателей и студентов

УДК 681.3.06
ББК 32.973.26-018.2

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зав. редакцией	*Анна Кузьмина*
Редактор	*Григорий Добин*
Компьютерная верстка	*Ольги Сергиенко*
Корректор	*Зинаида Дмитриева*
Оформление серии	*Via Design*
Дизайн обложки	*Игоря Цырульникова*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 22.11.02.
Формат 70×100$^{1}/_{16}$. Печать офсетная. Усл. печ. л. 61,92.
Тираж 4000 экз. Заказ №
"БХВ-Петербург", 198005, Санкт-Петербург, Измайловский пр., 29.

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

Отпечатано с готовых диапозитивов
в ФГУП ордена Трудового Красного Знамени "Техническая книга"
Министерства Российской Федерации по делам печати,
телерадиовещания и средств массовых коммуникаций.
198005, Санкт-Петербург, Измайловский пр., 29.

ISBN 5-94157-243-3

© Гарнаев А. Ю., 2002
© Оформление, издательство "БХВ-Петербург", 2002

Содержание

Предисловие **21**

О чем эта книга 21
Для кого предназначена эта книга 22
Структура книги 22
Типографские соглашения 25
Об авторе 26

Глава 1. Базовые сведения о VBA **27**

Что такое VBA 27
Что такое объект 28
    Семейство объектов 28
Создание функции пользователя в VBA 28
    Где пишется код функции пользователя 28
    Структура кода функции пользователя 29
    Ваша первая функция пользователя 30
    Вычисление стоимости партии продаваемых книг при помощи пользовательской функции 32
    Параметрами пользовательских функций могут быть ссылки на диапазоны 34
Автоматизация работы рабочего листа при помощи элементов управления 34
    Работа с макрорекордером 35
    Структура кода процедуры 37
    Процедура обработки события 38
    Элемент управления *Кнопка*, создание процедуры обработки события и автоматическое перелистывание рабочих листов 38
Построение шаблона таблицы 41
    Управление диаграммой 44
Создание скриптов 45
    Создание скриптов для Windows 45
    Создание скриптов для Web на стороне клиента 46
    Создание скриптов для Web на стороне сервера 47

Глава 2. Интегрированная среда разработки **50**

Где набирается код VBA 50
Структура редактора VBA 50
    Окно *Project — VBAProject* 51

Копирование модулей и форм из одного проекта в другой 52
Окно редактирования кода 52
Интеллектуальные возможности редактора кода 54
Окно *UserForm* (Редактирование форм) 56
Окно *Properties* (Свойства) 59
Окно *Object Browser* (Просмотр объектов) 60

Глава 3. Базовые объекты 62

Объектная модель 62
Полная и неявная ссылка на объект 62
Объект *Application* 67
Свойства объекта *Application* 67
Ссылка на активную рабочую книгу, лист, ячейку, диаграмму и принтер 68
Инсталлированные надстройки 68
Предупреждение перед копированием значения в заполненную ячейку 69
Отображение помощника при выборе ячейки 69
Автозамена 70
Замена текста гиперссылкой 72
Автосохранение 72
Управление уровнем безопасности 72
Задание режима вычисления 73
Форсирование вычислений перед закрытием рабочей книги 73
Проверка состояния вычисления 74
Управление перетаскиванием ячеек 74
Диапазон ячеек 74
Столбцы и строки рабочего листа 75
Листы с диаграммами 75
Рабочая книга, в которой выполняется данный макрос 76
Установка заголовка окна MS Excel 76
Установка курсора 77
Блокировка листа на время ввода данных 78
Каталог, используемый по умолчанию MS Excel при открытии файлов 78
Формат, используемый по умолчанию при сохранении рабочей книги 78
Установка опций книги, когда она используется как Web-страница 78
Семейство встроенных диалоговых окон 78
Отображение встроенных предупреждений о работе программы 79
Отображение примечаний 79
Отображение строки формул, полосы прокрутки и строки состояния 81
Задание текста, выводимого в строке состояния 81
Полноэкранное отображение рабочего листа 82
Управление действием операционной системы по нажатии комбинации клавиш <Ctrl>+<Break> 82
Установка высоты и ширины окна приложения 82
Установка координаты левого верхнего угла окна приложения 82
Максимальные размеры рабочей книги 83
Нахождение пути к файлу Excel.exe и к библиотечному файлу 84
Управление возможностью редактирования данных в ячейке 84

Поиск файлов ... 84
Свободная, общая и используемая память ... 86
Имя операционной системы ... 86
Название организации, для которой зарегистрирована операционная система, имя пользователя и код продукта ... 87
Семейство всех имен активной рабочей книги ... 87
Семейство недавно использованных файлов ... 87
Установка стиля ссылок на диапазоны ... 88
Обновление изображения во время выполнения программы ... 89
Ссылка на выбранный объект ... 89
Семейство листов активной рабочей книги ... 89
Печать активного листа ... 91
Предварительный просмотр книги ... 91
Доступ из кода к функциям рабочего листа ... 92
Номер версии MS Excel ... 92
Семейство всех окон приложения ... 92
Управление состоянием окна ... 93
Методы объекта *Application* ... 93
Запуск или активизация MS-приложения ... 93
Принудительное вычисление во всех открытых рабочих книгах ... 94
Перевод сантиметров и дюймов в пункты ... 94
Проверка правописания отдельного слова ... 94
Перевод ссылки на диапазон из формата *A1* в формат *R1C1* ... 95
Диалоговое окно ввода данных ... 96
Ввод ссылки на ячейку ... 97
Ввод числа ... 98
Преобразование имени MS Excel в объект или значение ... 98
Симулирование вычисления арифметических выражений ... 99
Простейший функциональный калькулятор ... 99
Симулирование ячеек рабочего листа ... 99
Применение квадратных скобок при симулировании операций на рабочем листе ... 100
Отображение справки ... 100
Перевычисление функции пользователя при изменениях в ячейках рабочего листа ... 101
Установка выполнения специфицированной процедуры при нажатии заданной комбинации клавиш ... 101
Назначение выполнения процедуры на определенное время ... 103
Электронные часы в ячейке рабочего листа ... 103
Электронный будильник ... 104
Отмена задания у часов ... 104
Закрытие приложения ... 105
Пересечение и объединение диапазонов ... 105
Проверка принадлежности ячейки выделенному диапазону ... 105
Запуск макроса или вызов функции ... 106
Сохранение изменений во всех рабочих книгах ... 107
Приостановка работы приложения до указанного времени ... 108
События объекта *Application* ... 108

Семейство *Workbooks* ... 110
Методы семейства *Workbooks* ... 111
Создание новой рабочей книги ... 111
Проверка того, можно ли редактировать книгу, расположенную на сервере ... 111
Редактирование книги, расположенной на сервере ... 112
Закрытие рабочей книги с сохранением и не с сохранением изменений ... 112
Открытие рабочей книги ... 113
Открытие текстового файла как новой рабочей книги ... 114
Открытие базы данных ... 115
Открытие XML-файла ... 116
Объект *Workbook* ... 116
Свойства объекта *Workbook* ... 116
Закрытие книги без сохранения изменений ... 120
Маршрутизация рассылки книги ... 121
Методы объекта *Workbook* ... 121
Переход к документу ... 123
Установка и снятие защиты книги ... 124
События объекта *Workbook* ... 125
Запрет закрытия рабочей книги ... 126
Обязательный пересчет данных в рабочей книге перед ее печатью ... 127
Обязательный запрос на подтверждение сохранения рабочей книги ... 127
Управление процессом создания нового листа ... 128
Семейство *Worksheets* ... 128
Объект *Worksheet* ... 130
Свойства объекта *Worksheet* ... 131
Очистка диапазона с данными ... 134
Запрет на выделение пользователем диапазонов ... 134
Проверка, установлена ли защита на содержании рабочего листа ... 135
Методы объекта *Worksheet* ... 135
Удаление рабочего листа без предупреждения пользователя ... 136
Смена фоновой картинки при щелчке правой кнопкой на ячейке ... 137
Установка и снятие защиты с рабочего листа ... 138
Объект *Protection* (как определить, какая защита на рабочем листе установлена) ... 139
События объекта *Worksheet* ... 140
Блокировка действий, связанных с событием по умолчанию ... 140
Автоматическое включение гиперссылки, содержащей ключевое слово, в папку *Избранное* ... 141
Автоматическое переоформление таблицы при изменении в ней значений ... 142
Объекты *Range* и *Selection* ... 143
Адресация ячеек ... 144
Задание групп строк и столбцов ... 145
Связь объекта *Range* и свойства *Cells* объекта *Worksheet* ... 145
Свойства объекта *Range* ... 146
Ввод или считывание значения из диапазона ... 146
Ввод в диапазон массива значений ... 146
Поиск по шаблону подобных значений в диапазоне ... 147

Ввод или считывание формулы в ячейку в формате *A1* 148
Ввод или считывание формулы в ячейку в формате *R1C1* 148
Ввод или считывание формулы локальной версии в ячейку в формате *A1* 148
Ввод или считывание формулы локальной версии в ячейку в формате *R1C1* 148
Ввод формулы массива в диапазон 148
Ввод формулы массива локальной версии в диапазон 149
Ввод формулы массива в диапазон с относительными ссылками на ячейки 149
Как узнать, спрятана ли формула на защищенном листе 149
Как узнать, имеется ли в ячейке формула 149
Определение адреса ячейки 149
Может ли ячейка быть редактирована на рабочем листе 150
Определения числа областей, из которых состоит данный диапазон 151
Управление стилем границы диапазона и объекта *Border* 152
Функции *RGB* и *QBColor* 153
Доступ к отдельным ячейкам диапазона 154
Объект *Characters* (как форматировать часть содержимого ячейки) 156
Номер первого столбца и строки первой области диапазона 157
Семейство всех столбцов и строк данного диапазона 157
Нахождение номера последнего столбца и строки диапазона 157
Установка ширины и высоты как самого диапазона, так и всех его строк и столбцов 158
Число ячеек диапазона 158
Объект *Comment* (создание комментариев к диапазону) 158
Определение текущего диапазона 160
Нахождение крайней ячейки диапазона в указанном направлении 161
Нахождение строки и столбца, содержащих данную ячейку 161
Как спрятать строку или столбец 161
Горизонтальное и вертикальное выравнивание содержимого диапазона 162
Объект *Hyperlink* (задание гиперссылки) 162
Объект *Font* (задание шрифта) 164
Объект *Interior* (заливка диапазона) 165
Отмена заливки диапазона 167
Местоположение диапазона и привязка расположения объекта к конкретной ячейке 167
Разрешение редактирования содержимого ячеек на защищенном рабочем листе 167
Имя диапазона 168
Установка числового формата 168
Нахождение диапазона, сдвинутого относительно данного на указанное число строк и столбцов 169
Задание угла, под которым выводится текст в диапазоне 169
Переопределение размеров диапазона 170
Автоматическое изменение размеров шрифта с тем, чтобы текст умещался в диапазоне 170
Установка стандартной высоты строки и ширины столбца 170
Автоматический перенос слов 171
Как узнать, на каком листе находится данная активная ячейка 171

Методы объекта *Range* ... 171
Активизация и выбор диапазона ... 172
Вставка и удаление комментариев в диапазон ... 172
Автоматический подбор размеров диапазона так, чтобы в нем помещались введенные данные ... 172
Заполнение диапазона прогрессией ... 173
Автозаполнение ячеек диапазона элементами последовательности ... 175
Табуляция функции ... 176
Заполнение диапазона по одному значению ... 177
Обрамление диапазона границей ... 178
Очиста ячейки ... 178
Копирование, вырезание и удаление данных из диапазона ... 179
Специальная вставка ... 179
Вставка диапазона с транспонированием ... 180
Снятие выделения после специальной вставки ... 181
Добавление ячейки, строки или столбца ... 181
Копирование диапазона в буфер обмена как растровое изображение ... 181
Отображение окна мастера функций ... 182
Поиск значения в диапазоне ... 182
Повторный поиск и поиск всех значений ... 184
Замена значений ... 185
Подбор параметра и решение уравнения с одним неизвестным ... 185
Прокрутка экрана с тем, чтобы диапазон отобразился на экране ... 187
Семейство *Windows* ... 187
Свойства семейства *Windows* ... 187
Упорядочивание окон ... 188
Объект *Window* ... 188
Свойства объекта *Window* ... 189
Установка местоположения и размеров окна ... 191
Методы объекта *Window* ... 191
Перемещение по иерархии объектов при помощи свойства *Parent* ... 192
Объект *Debug* и окно *Immediate* ... 193

Глава 4. Основы программирования на VBA ... 194

Допустимые имена ... 194
Типы данных ... 195
Переменные ... 196
Директива *Option Explicit* ... 198
Область видимости переменной ... 198
Время жизни переменной ... 199
Ключевое слово *Null* ... 199
Комментарии ... 199
Перенос строки кода ... 200
Расположение нескольких операторов в одной строке ... 200
Строки ... 200
Конкатенация строк ... 201
Ключевое слово *Empty* и пустая строка ... 201

Массивы ... 201
Изменение базового индекса ... 202
Поэлементная инициализация массива ... 203
Инициализация массива при помощи функции *Array* ... 203
Массив и диапазон ... 204
Динамические массивы ... 204
Определение границ индексов массива ... 206
Как проверить, содержит ли переменная типа *Variant* массив значений ... 206
Повторная инициализация массива и высвобождение памяти, выделенной под массив ... 207
Константы ... 208
Перечисляемый тип ... 208
Тип данных, определенный пользователем ... 209
Операции VBA ... 211
Математические операции ... 211
Операции отношения ... 212
Логические операции ... 212
Директива *Option Compare* ... 213
Приоритеты операций ... 214
Встроенные функции VBA ... 215
Математические функции ... 215
Создание последовательности случайных чисел ... 216
Повторение последовательности случайных чисел ... 216
Получение случайного числа из целочисленного интервала ... 217
Как найти значение числа π ... 217
Функции проверки типов ... 217
Преобразование строки в число ... 218
Преобразование числа в строку ... 218
Другие функции преобразования типов ... 219
Форматирование числа ... 220
Пользовательские форматы ... 222
Форматирование чисел ... 224
Форматирование процентов ... 224
Денежный формат ... 225
Форматирование даты и времени ... 225
Функции обработки строк ... 225
Нахождение ASCII-кода по литералу и литерала по ASCII-коду ... 226
Преобразование строки к нижнему или верхнему регистру ... 226
Возвращение подстроки по указанному местоположению ... 226
Определение длины строки ... 227
Копия строки без начальных и конечных пробелов ... 227
Строка, состоящая из указанного числа пробелов ... 227
Строка, состоящая из указанного числа повторяющихся символов ... 228
Преобразование строки в массив ... 228
Преобразование массива в строку ... 229
Сравнение строк ... 229
Создание зеркальной строки по отношению к данной ... 230
Нахождение вхождения в строку подстроки ... 230
Замена в строке подстроки ... 230

Функции, возвращающие строки ... 231
Функции времени и даты ... 231
Определение текущей системной даты ... 232
Определение текущего системного времени ... 232
Определение текущей даты и системного времени ... 232
Извлечение из времени часового, минутного и секундного компонентов ... 232
Извлечение из даты годового, месячного и дневного компонентов ... 233
Определение дня недели ... 233
Определение числа секунд, прошедших с полуночи ... 234
Определение количества лет, кварталов, месяцев, недель и дней, прошедших между двумя датами ... 234
Определение компонента даты ... 235
Добавление к дате указанного временного интервала ... 235
Преобразование часов, минут и секунд в формат времени ... 236
Преобразование года, месяца и дня в формат даты ... 236
Преобразование строки в формат времени ... 236
Функции выбора ... 237
Встроенные диалоговые окна ... 238
Окно ввода ... 238
Как обработать нажатие кнопки *Cancel* ... 240
Окно сообщения ... 240
Определение нажатой кнопки в окне ввода ... 243
Финансовые функции ... 244
Оператор присваивания ... 248
Присваивание переменной ссылки на объект ... 249
Оператор *With* ... 249
Операторы управления ... 250
Оператор условного перехода ... 250
Оператор выбора ... 252
Оператор *For Next* ... 254
Оператор *For Each* ... 255
Использование оператора *For Each* при суммировании элементов массива ... 256
Использование оператора *For Each* при переборе ячеек диапазона ... 256
Использование оператора *For Each* при переборе элементов семейства ... 257
Оператор *While* ... 257
Создание бесконечного цикла оператором *While* ... 258
Оператор *Do* ... 258
Альтернативный выход из цикла ... 260
Создание бесконечного цикла оператором *Do* ... 260
Оператор безусловного перехода *GoTo* ... 260
Процедура ... 261
Список параметров процедуры ... 263
Организация программы на языке VBA ... 264
Вызов процедуры и передача значений параметров ... 265
Процедура с необязательными параметрами ... 266
Специфицирование значений по умолчанию необязательным параметрам ... 267
Использование неопределенного количества параметров ... 267

Использование массива в качестве параметра процедуры ... 268
Передача параметров по ссылке и значению ... 269
Рекурсивные процедуры ... 269
Фракталы ... 270
Используя рекурсию, будьте осторожны ... 274

Глава 5. Классы, объекты и семейства ... 275

Классы и экземпляры ... 275
Объявление класса ... 276
Создание экземпляра класса ... 277
Инициализация значений полей ... 277
Ключевое слово *Me* ... 278
Ключевое слово *Nothing* и удаление объекта из памяти ... 278
Методы ... 278
Свойства как средство ограничения доступа к полям класса ... 280
Свойства "только для чтения" и "только для записи" ... 282
События ... 283
Таймер как пример класса, генерирующего события ... 285
Объект *Collection* ... 287

Глава 6. Элементы управления на рабочем листе ... 288

Панель инструментов *Элементы управления* ... 288
Размещение элемента управления на рабочем листе и написание кода ... 289
Ваш первый проект с элементом управления ... 291
Общие свойства элементов управления ... 293
Общие методы элементов управления ... 294
Общие события элементов управления ... 295
Кнопка (*CommandButton*) ... 296
Кнопочное меню ... 296
Навигация по книге при помощи гиперссылок ... 297
Кнопочный сценарий ... 298
Кнопочный сценарий для ввода формул с кнопками, украшенными рисунками, и пользовательским указателем мыши ... 300
Интерактивная кнопка и определение среднего объема продаж ... 303
Обмен значений между двумя выбранными ячейками ... 306
Переключатель (*OptionButton*) ... 307
Переключатели и объемы продаж ... 307
Флажок (*CheckBox*) и Выключатель (*ToggleButton*) ... 309
Флажок и управление отображением элементов диаграммы ... 310
Выключатель и отображение примечаний ... 311
Полоса прокрутки (*ScrollBar*) и Счетчик (*SpinButton*) ... 313
Ввод значений в ячейку и управление цветом ... 314
Ввод в ячейку с помощью полосы прокрутки и счетчика нецелочисленных значений ... 316
Список (*ListBox*) ... 318
Сценарии со списком ... 320
Защита ячеек рабочего листа ... 321
Управление печатью элементов управления ... 322

Глава 7. Форма и элементы управления 325

Форма 325
Добавление формы в проект 325
Семейство форм 325
Свойства формы 326
Методы формы 328
События формы 329
Отображение и скрытие формы 330
Первый проект с формой 330
Как запустить проект на исполнение 332
Ключевое слово *Me* 332
Форма с обновляемым фоновым рисунком 332
Удаление рисунка 334
Форма с мозаичным фоном и установкой свойств на этапе инициализации 334
Закрытие формы при нажатии клавиши <Esc> 336
Подтверждение на закрытие окна 336
Задание местоположения формы 337
Модальная форма 338
Использование нескольких форм 338
"Пасхальное яйцо" 339
Элементы управления 340
Размещение элемента управления на форме 342
Надпись (*Label*) 342
Поле (*TextBox*) 343
Сложение двух чисел 344
Кнопка с "горячей" клавишей 346
Клавиши <Enter> и <Esc> 346
Суммирование с блокировкой результата для пользователя 347
Как сделать, чтобы по нажатии на кнопке она не получала фокус 347
Перемещение фокуса между полями при нажатии клавиши <Enter> 347
Всплывающая подсказка 348
Поле вода пароля 349
Многострочное поле ввода 350
Обмен значениями между формами 352
Флажок (*CheckBox*) и Выключатель (*ToggleButton*) 353
Управление видимостью элементов управления 353
Управление достижимостью для пользователя элементов управления 354
Рамка (*Frame*) 355
Переключатель 355
Переключатель и выбор результирующей операции 355
Полоса прокрутки (*ScrollBar*) и Счетчик (*SpinButton*) 357
Синхронизированная работа поля ввода и счетчика 357
Усовершенствование элемента *Подбор параметра* 359
Список (*ListBox*) 362
Поэлементное заполнение списка 362
Заполнение списка из массива и выбор операции 364
Заполнение списка из диапазона 365

Выбор нескольких элементов из списка ... 367
Согласованная работа двух списков ... 369
Многостолбцовый список ... 370
Заполнение многостолбцового списка из диапазона и нахождение среднего значения выбранных чисел ... 372
Скрытие данных в многостолбцовом списке ... 374
Вывод в многостолбцовом списке выбранного значения при помощи свойств *Text* и *Value* ... 376
Буксировка элементов из одного списка в другой ... 377
Поле со списком (*ComboBox*) ... 379
Поле со списком, ввод данных в алфавитном порядке и объект *Collection* ... 379
Добавление и удаление данных в поле со списком ... 382
Рисунок (*Image*) ... 384
Окно *О программе* ... 385
Просмотр слайдов ... 386
Модифицированный мастер диаграмм ... 387
Элемент управления *RefEdit* ... 391
Определение статистических параметров диапазона ... 391
Решение системы линейных уравнений ... 392
Набор страниц (*MultiPage*) и набор вкладок (*TabStrip*) ... 394
Статистика и набор страниц ... 395
Последовательность перехода элементов управления ... 397
Отображение встроенных диалоговых окон ... 398
Открытие документа и метод *GetOpenFilename* ... 400
Простейший браузер для графических файлов ... 401
Сохранение документа и метод *GetSaveAsFilename* ... 402
Дополнительные элементы управления ... 403
Добавление дополнительного элемента управления ... 403
Удаление дополнительного элемента управления ... 404
Пример дополнительного элемента управления — *Календарь* (Calendar) ... 404
Заполнение табличной базы данных ... 407

Глава 8. Диаграммы ... 413

Построение шаблона отчета с диаграммой ... 413
Семейства *ChartObjects* и *Charts* и объекты *ChartObject* и *Chart* ... 418
Добавление нового элемента в семейства *ChartObjects* и *Charts* ... 419
Свойства объекта *Chart* ... 420
Методы объекта *Chart* ... 423
События объекта *Chart* ... 424
Метод *ChartWizard* ... 424
Построение диаграммы ... 425
Изменение диапазона, по которому строится диаграмма ... 428
Изменение типа диаграммы ... 431
Автоматическое определение диапазона, по которому строится диаграмма ... 433
Линия тренда ... 434
Проект с линией тренда ... 435
Построение поверхности и управление ее ориентацией ... 439

Установка защиты на вложенную в рабочий лист диаграмму ... 442
Защита диаграммы, расположенной на отдельном листе ... 443
События и диаграммы ... 444
Привязка событий к вложенным в рабочий лист диаграммам ... 445
Изменение типа диаграммы при помощи контекстного меню ... 446

Глава 9. Создание меню, контекстного меню и панели инструментов ... 449

Семейство *CommandBars* ... 449
Объект *CommandBar* ... 451
Встроенные панели инструментов ... 452
Пункты меню или элементы управления панели инструментов и семейство *CommandBarControls* ... 453
Пункт меню или кнопка панели инструментов и объект *CommandBarButton* ... 454
Создание строки меню ... 456
Создание сложного меню ... 459
Пункты меню с картинками и со встроенными функциями ... 461
Скрытие и отображение панелей инструментов ... 464
Создание панели инструментов с кнопками и защитой от изменений со стороны пользователя ... 465
Массив кнопок в панели инструментов ... 468
Раскрывающийся список панели инструментов и объект *CommandBarComboBox* ... 469
Панель инструментов с раскрывающимся списком ... 471
Событие, связанное с раскрывающимся списком ... 473
Конструирование контекстного меню ... 474

Глава 10. Обработка данных: консолидация, сортировка, сценарии, фильтрация, сводные таблицы, поиск решения ... 477

Консолидация данных ... 477
Консолидация при помощи трехмерных формул на рабочем листе ... 478
Консолидация при помощи трехмерных формул в коде ... 479
Консолидация данных по положению и категориям ... 480
Методы и свойства, используемые при программировании консолидирующей таблицы ... 482
Пример приложения, консолидирующего данные ... 484
Сортировка данных ... 487
Сортировка по выделенному полю ... 487
Сортировка выделенной области ... 488
Сортировка по двум или трем столбцам ... 488
Сортировка по списку или по дням недели ... 489
Сортировка с учетом регистра ... 489
Восстановление первоначального порядка вывода данных ... 490
Сортировка на защищенном листе ... 490
Метод *Sort* ... 490
Пример приложения, сортирующего данные ... 492
Фильтрация ... 494
Использование автофильтра ... 495
Описание процесса автофильтрации ... 495

Метод *AutoFilter* ... 497
Отображение и удаление кнопок со стрелками автофильтра ... 498
Отображение всех отфильтрованных данных ... 498
Как узнать, что на листе имеются кнопки со стрелками автофильтра ... 498
Как узнать, что на листе имеются отфильтрованные данные ... 498
Объекты *AutoFilter* и *Filter* и семейство *Filters* ... 498
Доступ к раскрывающимся спискам автофильтра на защищенном листе ... 500
Пример приложения, фильтрующего данные ... 500
Промежуточные итоги ... 503
Описание процесса создания промежуточных итогов ... 504
Метод *Subtotal* ... 506
Удаление промежуточных итогов ... 507
Структура и объект *Outline* ... 508
Отображение указанного числа уровней структуры ... 508
Удаление структуры ... 509
Отображение значков структуры ... 509
Автоматическое создание структуры ... 509
Пример приложения, подводящего промежуточные итоги и управляющего структурой ... 509
Сценарии ... 511
Объект *Scenario* ... 512
Пример приложения по работе со сценариями ... 513
Сценарии без объекта *Scenario* ... 515
Сводная таблица ... 516
Подведение итогов по клиентам ... 516
Обновление данных ... 523
Подведение итогов по клиентам и странам ... 524
Сортировка данных в сводной таблице ... 525
Отображение заданного числа лучших или худших результатов ... 525
Группировка данных по дням, месяцам, кварталам и годам ... 526
Удаление группировки ... 529
Удаление сводной таблицы ... 529
Нахождение как суммарной стоимости заказов каждого клиента, так и той доли, которую они занимают по отношению ко всем клиентам ... 529
Отображение детальных данных, содержащихся в ячейке сводной таблицы ... 530
Объекты, связанные со сводной таблицей ... 531
Метод *PivotTableWizard* ... 531
Объект *PivotTable* ... 533
Объект *PivotCache* ... 534
Объект *PivotField* ... 535
Пример приложения, помогающего построить и обновлять сводную таблицу ... 536
Поиск решения ... 541
Решение транспортной задачи при помощи средства Поиск решения ... 542
Функции, программирующие поиск решения ... 548
Приложение Транспортная задача ... 551
Решение оптимизационных задач, зависящих от параметра ... 555
Решение уравнения, зависящего от двух параметров ... 558

Глава 11. Обработка ошибок и отладка программ 561

Разработка процедур, предотвращающих появление ошибок 561
Контроль вводимых значений с помощью обработки события *KeyPress* 564
Перехват и обработка ошибок 566
Оператор *On Error* 567
Процедура обработки ошибки 567
Оператор *Resume* 567
Объект *Err* 568
Перехват ошибок с предложением пользователю выбора действий 571
Построчный перехват ошибок 573
Отладка программ 574
Ошибки компиляции 574
Ошибки выполнения 576
Логические ошибки 578
Инструкция *Option Explicit* 578
Пошаговое выполнение программ 579
Точка прерывания 580
Вывод значений свойств и переменных 581
Окно *Watches* 581
Окно *Locals* 582
Окно *Immediate* 582
Программный способ вывода значений в окно *Immediate* 582

Глава 12. Помощник MS Office 583

Объект *Assistant* 583
Свойства объекта *Assistant* 583
Методы объекта *Assistant* 585
Объект *Balloon* 586
Помощник с надписями и вызовом процедуры 588
Помощник с флажками 590
Добавление значка в заголовок помощника 591
Добавление значка в надписи и флажки помощника 593
Раскрашивание помощника 595

Глава 13. Объектная файловая система 596

Объект *FileSystemObject* и доступ к объектной файловой системе 596
Семейства *Drives, Folders* и *Files* 598
Объект *Drive* 598
Как проверить, существует ли диск 599
Как проверить, доступен ли диск 599
Получение информации о диске 600
Объект *Folder* 601
Как проверить, существует ли каталог 602
Создание каталога 603
Копирование каталога 603
Перемещение каталога 604
Удаление каталога 605

Получение информации о каталоге ... 606
Определение уровня данного каталога ... 607
Список подкаталогов данного каталога ... 608
Объект *File* ... 609
Как проверить, существует ли файл ... 610
Копирование и перемещение файла ... 610
Удаление файла ... 611
Информация о файле ... 612
Изменение атрибутов файла ... 613
Список всех файлов данного каталога ... 614
Объект *TextStream* ... 614
Открытие файла и создание объекта *TextStream* ... 615
Метод *OpenAsTextStream* ... 615
Метод *CreateTextFile* ... 617
Метод *OpenTextFile* ... 618
Ввод и вывод данных из поля ввода в текстовый файл ... 619
Запись, присоединение и считывание данных из файла ... 620
Запись данных в HTML-файл ... 621
Объект *Dictionary* ... 623
Создание адресной книги с помощью объекта *Dictionary* ... 623
Функции по работе с файлами ... 625
Поиск файла и объект *FileSearch* ... 626

Глава 14. Базы данных ... 629

Ключи и индексы ... 631
База данных *Борей* ... 632
Отношения ... 634
Создание базы данных в MS Access ... 634
Создание таблицы базы данных в MS Access ... 635
Создание отношений с помощью MS Access ... 639
Запрос и язык запросов SQL ... 641
Оператор *SELECT* ... 641
Возвращение всех записей таблицы ... 642
Возвращение всех записей одного поля ... 643
Возвращение всех записей двух полей таблицы ... 643
Упорядочивание элементов и предложение *ORDER BY* ... 643
Упорядочивание записей по одному полю ... 643
Упорядочивание записей по двум полям ... 643
Выборка записей по критерию и предложение *WHERE* ... 643
Ссылка на строки и даты в операциях сравнения ... 644
Предложение *WHERE* и выборка записей по простому критерию ... 644
Выборка записей по составному критерию ... 644
Выборка записей, значения специфицированного поля которых принадлежат указанному диапазону или лежат за его пределами ... 644
Выборка записей, значения специфицированного поля которых принадлежат указанному множеству ... 645
Выборка записей по шаблону ... 645
Выбор различных записей ... 645

Нахождение общего числа записей, возвращаемого запросом ... 645
Нахождение общего числа непустых записей ... 645
Статистические функции ... 646
Определение, пусто ли поле, и значение *NULL* ... 646
Оператор *DELETE* ... 647
Оператор *INSERT* ... 647
Оператор *UPDATE* ... 647
Создание групп записей ... 647
Создание псевдонима с использованием предложения *AS* ... 648
Предложение *HAVING* ... 648

Глава 15. ADO (ActiveX Data Objects) ... 649

Создание ссылки на библиотеку ADO ... 650
Объект *Connection* и установка подключения к базе данных ... 651
Установка DSN ... 652
Объект *Recordset* и его создание ... 652
Использование свойств *BOF* и *EOF* при навигации по записям ... 653
Пример использования объекта *Recordset* для просмотра базы данных ... 654
Создание объекта *Recordset* без предварительного создания объекта *Connection* ... 655
Методы, свойства и события объекта *Recordset* ... 656
Определение числа элементов в наборе записей ... 659
Последовательный просмотр записей базы данных ... 660
Восстановление объекта *Recordset* с помощью методов *Close* и *Open* ... 662
Работа с курсорами ... 663
Объект *Field* ... 663
Как при вводе данных в поле ввода из пустой ячейки таблицы избежать генерации ошибки ... 664
Создание браузера базы данных ... 665
Браузер просмотра нескольких таблиц базы данных ... 667
Сохранение и открытие набора записей файлового типа ... 670
Редактирование, создание, обновление и удаление записей ... 672
Создание закладок ... 677
Поиск записи методом *Find* ... 679
Как избежать написания большого числа апострофов при построении критерия поиска ... 682
Применение баз данных при создании интернациональных приложений ... 683
Как сохранить набор записей в XML-формате ... 686
Вывод набора записей на рабочий лист ... 687
Использование объекта *Command* ... 688

Глава 16. Интеграция приложений ... 690

Технология ActiveX ... 690
Связывание и внедрение объектов ... 691
Внедрение нового объекта ... 691
Внедрение существующего объекта ... 692
Технология Automation ... 692
Программные идентификаторы приложений-серверов Automation ... 693

Функции доступа к объектам Automation ... 694
Позднее и ранее связывание ... 695
Microsoft Excel и Microsoft Word ... 696
Создание нового документа Microsoft Word функцией *CreateObject* ... 696
Открытие документа Microsoft Word функцией *GetObject* ... 697
Создание отчета из MS Excel в MS Word ... 698
MS Excel и MS Access ... 702
Создание объектов MS Access ... 702
Печать или предварительный просмотр отчета ... 703
Экспорт данных из MS Access в MS Excel ... 704
Создание презентации в MS PowerPoint ... 706

Глава 17. MS Excel и Internet ... 708

Гиперссылки ... 708
Добавление гиперссылок на документы MS Office ... 709
Задание гиперссылки формулой рабочего листа ... 710
Условная гиперссылка ... 711
Объект *Hyperlink* и семейство *Hyperlinks* ... 711
Переход по гиперссылке из списка ... 713
Web-запрос и получение данных с Web-страницы ... 714
Создание Web-запроса ... 714
Создание скриптов для Web на стороне сервера ... 716
Передача данных от клиента серверу ... 718
Создание калькулятора на базе MS Excel в клиент-серверных приложениях ... 720
Создание Web-страницы для доступа к базе данных, расположенной на сервере ... 722

Глава 18. MS Excel и Visual Basic .NET ... 724

Открытие и создание рабочей книги ... 724
Нахождение значений арифметических выражений ... 725
Проверка орфографии ... 727
Решение уравнения ... 728
Построение графика ... 733
Создание отчета на рабочем листе ... 737

Глава 19. MS Excel и XML ... 740

Просмотр XML-файлов в MS Excel ... 740
Синтаксис XML ... 741
Русификация XML ... 744
Применение каскадной таблицы стилей для форматного вывода текста ... 744
Комментарии в XML ... 745
Что такое DTD ... 745
Внутренний блок DTD ... 747
Элемент ... 747
Задание частоты и альтернативы использования элемента ... 747
Объявление атрибутов элемента ... 748
Как сохранить набор записей в XML-формате ... 749

Глава 20. Справочная система 751

Этапы построения справочной системы 751
Создание разделов справочной системы 751
Построение таблицы содержания 752
Конструирование файла проекта 754
Создание индекса 759
Подключение справки 759
Создание контекстно-зависимой справки 760
Подключение контекстной справки 760
Подключение контекстно-зависимой справки с использованием Помощника Microsoft Office 761
Подключение контекстно-зависимой справки при помощи панели инструментов 763

Предисловие

О чем эта книга

Microsoft Excel — это продукт, который поставляется в составе Microsoft Office и используется миллионами людей, как разработчиками, так и простыми пользователями. По некоторым оценкам, те пользователи (а их около 90% от общего числа), которые применяют только средства рабочего листа MS Excel, без использования возможностей, предоставляемых VBA (Visual Basic for Applications), реализуют лишь не более 10% от истинных способностей MS Excel. Таким образом, большая часть средств и возможностей MS Excel остается невостребованной просто потому, что пользователи не знают о наличии VBA или не умеют им пользоваться. Эта книга научит вас на все 100% использовать объединенную мощь MS Excel и VBA при разработке ваших приложений и сделает вас искусным разработчиком.

Целью книги является научить (и она с гарантией это сделает!) создавать профессиональные офисные проекты на основе MS Excel. В ней досконально разобрана объектная модель MS Excel, описан объектно-ориентированный подход к программированию офисных приложений, рассмотрены вопросы создания пользовательского интерфейса и справочной системы, автоматизации операций с рабочим листом и диаграммами, в том числе и при обработке и анализе данных и принятии решений. Подробно изложены средства работы с файлами, методы интеграции офисных приложений, работа с Internet, применение XML и использование MS Excel в клиент-серверных приложениях. Продемонстрированы возможности MS Excel по работе с базами данных средствами ADO и в проектах Visual Basic .NET.

Для большей наглядности и легкости усвоения материала в книге приводится около 400 примеров тщательно разработанных приложений: от создания пользовательских функций до построения информационных систем по сбо-

ру и обработке данных. Код в любом разделе самодостаточен и может быть непосредственно использован читателем при разработке собственных проектов.

Несмотря на то, что книга уделяет огромное внимание программированию на языке VBA, это совсем не требует от читателя быть профессиональным программистом. VBA обладает мощными встроенными интеллектуальными средствами, которые позволяют даже начинающему пользователю быстро самостоятельно разрабатывать профессиональные приложения. Так, при написании кода программы редактор VBA сам предлагает пользователю возможные продолжения составляемого им кода. Другим примером встроенных интеллектуальных средств VBA является макрорекордер, который переводит все вручную выполняемые пользователем на рабочем листе MS Excel действия на язык VBA. Таким образом, макрорекордер позволяет пользователю поручать компьютеру самому создавать большие куски кода разрабатываемого приложения.

Для кого предназначена эта книга

В книге предложен углубленный подход к созданию полнофункциональных информационных систем на основе MS Excel и VBA. Наличие некоторого опыта разработки программ и работы с электронными таблицами у читателя желательно, но совсем не обязательно. Книга написана по типу "вещь в себе", т. е. в ней приводятся описания всех используемых средств.

Таким образом, книга адресована как начинающим пользователям для самостоятельного изучения VBA и MS Excel, так и профессионалам в качестве справочника по самому языку, его объектной модели и современным приемам программирования, используемым при создании приложений. В книге можно найти ответы на типовые вопросы, с которыми сталкивается разработчик при создании офисных проектов.

Благодаря систематичности и структурированности изложения и наличию большого количества примеров книгу также можно использовать в качестве учебного пособия при преподавании VBA и MS Excel.

Структура книги

Книга состоит из 20 глав. Кратко опишем, о чем пойдет разговор в каждой из них.

В *первой главе* содержатся базовые сведения о VBA. Объясняется, в чем состоит суть VBA. Даны многочисленные примеры построения пользовательских функций на основе VBA: от простейших до сложных. Приводятся пер-

вые примеры приложений по автоматизации работы рабочего листа при помощи элементов управления. Объясняется, как посредством макрорекордера легко создается код приложений. Демонстрируется, как можно создавать Windows- и клиент-серверные сценарии.

Вторая глава посвящена описанию интегрированной среды разработки приложений VBA. В *третьей главе* обсуждаются базовые элементы объектной модели MS Excel: `Application`, `Workbook`, `Worksheet`, `Range`, `Window`, инкапсулирующие в себе данные о самом приложении, рабочей книге, диапазоне и окне. Кроме того, обсуждаются дочерние по отношению к данным объекты типа `Border` (граница), `Hyperlink` (гиперссылка), `Interior` (фон диапазона), `Font` (шрифт), `Protection` (данные об установленной защите), `Debug` (окно отладки).

Четвертая глава может быть рассмотрена как справочная, являющаяся полным руководством по основам программирования на VBA. В ней, в частности, описаны все типы данных, форматов, операций и операторов языка, приведены все встроенные функции VBA, включая финансовые, встроенные диалоговые окна, а также описана структура процедур VBA. Объяснено, как создаются процедуры с необязательными параметрами, с неопределенным количеством параметров, как по умолчанию назначаются значения параметрам. Глава насыщена огромным числом примеров поясняющего характера.

В VBA наряду с огромным числом встроенных объектов (`Application`, `Workbook`, `Worksheet`, `Range` и т. д.) предусмотрена возможность создания пользовательских объектов. Применение пользовательских объектов позволяет сократить текст программ, сделать их более прозрачными и понятными. Пользовательские объекты в VBA обладают свойствами, полями, методами, могут прослушивать события, но, к сожалению, не обладают привычным для объектно-ориентированных языков механизмом наследования. *Пятая глава* как раз и рассказывает о том, как можно создавать пользовательские объекты.

Шестая глава посвящена элементам управления, размещенным на рабочем листе, созданию индивидуального интерфейса проекта, внедренного непосредственно в рабочий лист, т. е. интерфейса, который максимально приближен к пользователю. Этот интерфейс позволяет автоматизировать и алгоритмизировать работу пользователя, тем самым упрощая ее и повышая ее эффективность. Элементы управления обеспечивают создание защиты "от дурака", т. е. с их помощью можно достичь того, чтобы пользователь мог действовать только в рамках бизнес-логики проекта, не нарушая его целостности.

Седьмая глава описывает формы. По своей сути форма (или пользовательская форма) представляет собой диалоговое окно, в котором можно размещать различные элементы управления. В приложении может быть как одна,

так и несколько форм. Используя комбинацию форм и элементов управления, возможно создать любой пользовательский интерфейс, максимально отображающий бизнес-логику проекта и предоставляющий оптимальные удобства пользователю.

MS Excel предоставляет в распоряжение разработчика мощное и гибкое средство — мастер диаграмм. В его арсенале имеется несколько десятков шаблонов плоских и объемных диаграмм, способных удовлетворить любые потребности пользователя для представления данных в наиболее доступном и наглядном графическом виде. *Восьмая глава* как раз и посвящена демонстрации возможностей мастера диаграмм, а также объектов `Chart` и `ChartObject`, которые позволяют автоматизировать как процесс построения диаграмм, так и их настройки в зависимости от бизнес-логики разрабатываемого проекта.

Девятая глава рассказывает о том, как создаются собственные панели инструментов, строки меню и контекстное меню, а также как исключаются из интерфейса ненужные по бизнес-логике проекта панели инструментов.

Десятая глава посвящена описанию эффективных средств по обработке и анализу данных, а именно по их консолидации, сортировке, фильтрации, подведению промежуточных итогов, конструированию сценариев и структур, а также созданию сводных таблиц и диаграмм. Встроенный в MS Excel мощный инструмент — Поиск решения — позволяет с легкостью решать оптимизационные задачи как линейного, так и нелинейного программирования. Применение же соответствующих объектных моделей обеспечивает создание полностью автоматизированных проектов как по обработке и анализу данных, так и принятию решения.

В *одиннадцатой главе* рассказывается о том, как в VBA обрабатываются и перехватываются ошибки, а также описаны средства отладки программ. *Двенадцатая глава* посвящена конструированию помощника MS Office, удобного средства по работе с офисными приложениями.

Тринадцатая глава рассказывает о работе с файлами, каталогами и дисками средствами объектной файловой системы (File System Object Model, FSO). FSO предоставляет разработчику возможность работы с файлами в ясной объектно-ориентированной манере. FSO опирается на библиотеку Microsoft Scripting Runtime Library (файл Scrrun.dll), которая совместно используется несколькими средами разработки и позволяет, в частности, расширить функции написания сценариев. В заключение главы будет рассмотрен объект `FileSearch` и ассоциированные с ним объекты, предоставляющие удобные средства поиска файлов на диске.

В *четырнадцатой главе* даны основные понятия баз данных, базовые сведения о построении SQL-запросов, а также демонстрируется, как можно создавать базы данных средствами MS Access.

Пятнадцатая глава посвящена технологии ADO (ActiveX Data Objects), предоставляющей разработчику универсальный метод извлечения и изменения данных из разнообразных источников, которые не ограничиваются только традиционными базами данных. Ими могут быть и файловая система, и Internet. Рука об руку с ADO идет OLE DB — набор интерфейсов COM (Component Object Model, компонентная модель объектов), используемых для извлечения данных. OLE DB предоставляет низкоуровневое подключение к данным через интерфейс COM, в то время как ADO предоставляет модель объектов, которая упрощает процесс использования OLE DB и написания кода. Технология ADO позволяет решать следующие задачи: создавать соединение с источником данных, создавать объект, реализующий SQL-команду, выполнять SQL-команду, сохранять результат выполнения SQL-команды в кэше, сконструировать виртуальное представление кэша, чтобы пользователь мог фильтровать, сортировать данные и перемещаться по ним, редактировать данные, обновлять источник данных в соответствии с внесенными изменениями, фиксировать или отменять изменения, внесенные в процессе транзакции и последующего ее закрытия.

В *шестнадцатой главе* описывается мощная технология Automation, которая существенно упрощает процесс и сокращает время разработки проектов. Технология Automation предоставляет разработчику доступ к объектам и методам других приложений, которые максимально приспособлены для решения специализированных задач. Теперь разработчику нет нужды каждый раз "изобретать велосипед", а достаточно просто воспользоваться технологией Automation. В данной главе демонстрируется, как можно интегрировать работу MS Excel с MS Word, MS Access и MS Powerpoint.

Семнадцатая глава рассказывает о связи MS Excel и Internet, о гиперссылках, о создании Web-запросов, о применении MS Excel в клиент-серверных приложениях.

В *восемнадцатой главе* даны примеры того, как в проекты Visual Basic .NET можно инкапсулировать возможности MS Excel при разработке приложений, а именно при создании калькуляторов, решении уравнений, построении графиков, конструировании отчетных таблиц.

Девятнадцатая глава описывает связь MS Excel и XML, а *двадцатая* — то, как можно конструировать справочную систему средствами HTML Help Workshop.

Типографские соглашения

Способ отображения	Применение
Полужирное начертание	Команды меню, кнопки панелей инструментов, заголовки и вкладки диалоговых окон Visual Basic

(окончание)

Способ отображения	Применение
`Шрифт Courier`	Опции и параметры диалоговых окон, методы, события, объекты и их свойства, листинги и фрагменты кода, составленные на языке Visual Basic
`[Параметр]`	При описании синтаксиса параметр, заключенный в квадратные скобки, является необязательным
`{Параметр1 | Параметр2}`	Если при описании синтаксиса несколько параметров заключены в фигурные скобки и разделителем между ними является вертикальная черта, то эти параметры являются альтернативными, и предполагается использование только одного из них
↳	Если при описании синтаксиса в начале строки расположен указанный символ, то это говорит о том, что данная строка является продолжением предыдущей
<Ctrl>+<R>	Знак плюс между названиями клавиш обозначает комбинацию клавиш (не отпуская первой клавиши, должна быть нажата вторая)

Об авторе

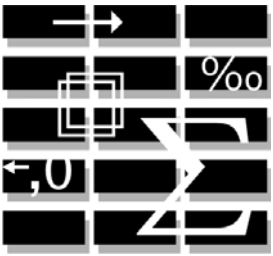
Андрей Юрьевич Гарнаев (garnaev@ag2784.spb.edu), доктор физико-математических наук, профессор Санкт-Петербургского государственного университета.

Область интересов: программное обеспечение, математическая кибернетика. Им опубликовано восемь книг по программированию и программным средствам, две — по теории игр. По офисному программированию он является автором книг "Excel 2000: разработка приложений", "Excel, VBA и Internet в экономике и финансах", "Visual Basic .NET: разработка приложений", вышедших в издательстве "БХВ-Петербург".

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

Автор благодарит вас за выбор этой книги и надеется, что ее чтение для вас будет не только полезным, но и увлекательным.

ГЛАВА 1



Базовые сведения о VBA

Что такое VBA

VBA — относительно легкий язык программирования. Он прост в освоении и позволяет быстро получать ощутимые результаты — конструировать профессиональные приложения, решающие практически все задачи, встречающиеся в среде Windows, а также при создании клиент-серверных приложений. При этом создание многих приложений с использованием VBA проще и быстрее, чем с применением других языков программирования.

VBA использует технологию визуального программирования, т. е. конструирование рабочей поверхности приложения и элементов его управления непосредственно на экране, а также запись всей программы или ее частей при помощи макрорекордера.

VBA позволяет создавать универсальные, автоматизированные приложения на платформе MS Excel. Кроме того, используя технологию ActiveX, VBA позволяет как внедрять в разрабатываемое приложение объекты других приложений, так и, не выходя из созданного приложения, управлять другими Windows-приложениями. В каком-то смысле VBA — это воплощение на компьютерном уровне идеи объединенной Европы.

VBA позволяет создавать клиент-серверные приложения, связывая ваш компьютер со всем остальным миром.

VBA относится к языкам объектно-ориентированного программирования (ООП). ООП можно описать как методику анализа, проектирования и написания приложений с применением объектов.

Прежде чем переходить к конкретным примерам использования VBA, давайте вкратце узнаем, что обозначают основные парадигмы ООП: объект, свойство, метод, событие, класс и семейство объектов.

Что такое объект

Объект — это парадигма ООП. Объект — это инкапсуляция данных вместе с кодом, предназначенным для их обработки, т. е. объединение данных и кода в нечто целое, именуемое объектом. Стоит сразу же отметить, что VBA не является объектно-ориентированным языком программирования в строгом понимании этого слова, однако объектный подход играет в нем основную роль. Все визуальные объекты, такие как рабочий лист (`Worksheet`), диапазон (`Range`), диаграмма (`Chart`), форма (`UserForm`), являются объектами. В VBA имеется более 100 встроенных объектов.

Семейство объектов

Семейство (объект `Collection`) представляет собой объект, содержащий несколько других объектов, как правило, одного и того же типа. Например, объект `Workbooks` содержит все открытые объекты `Workbook` (рабочая книга). Каждый элемент семейства нумеруется и может быть идентифицирован либо по номеру, либо по имени. Например, `Worksheets(1)` обозначает первый рабочий лист активной книги, а `Worksheets("Лист1")` — рабочий лист с именем **Лист1**.

Создание функции пользователя в VBA

В данном разделе на нескольких примерах покажем, как строятся функции пользователя. Начнем с простейшего примера — функции, вычисляющей значение по одной формуле. Затем обсудим более сложный пример, а именно расчет стоимости партии книг, когда значение функции зависит от условия. Следующий пример показывает, как создаются пользовательские функции, значениями параметров которых являются ссылки не на ячейку, а на диапазон. В последнем примере — расчета уравнения корреляции для повторяющихся данных наблюдений, показано, как конструируются пользовательские функции, возвращающие не отдельное значение, а целый диапазон значений.

Где пишется код функции пользователя

Сначала спросим себя, где строятся пользовательские функции. Ответ: в стандартном модуле редактора VBA. Для того чтобы попасть в редактор VBA, выберите в меню MS Excel команду **Сервис | Макрос | Редактор Visual Basic** или нажмите комбинацию клавиш <Alt>+<F11>. В результате вы попадете в интегрированную среду разработки приложений IDE редактора Visual Basic (рис. 1.1).

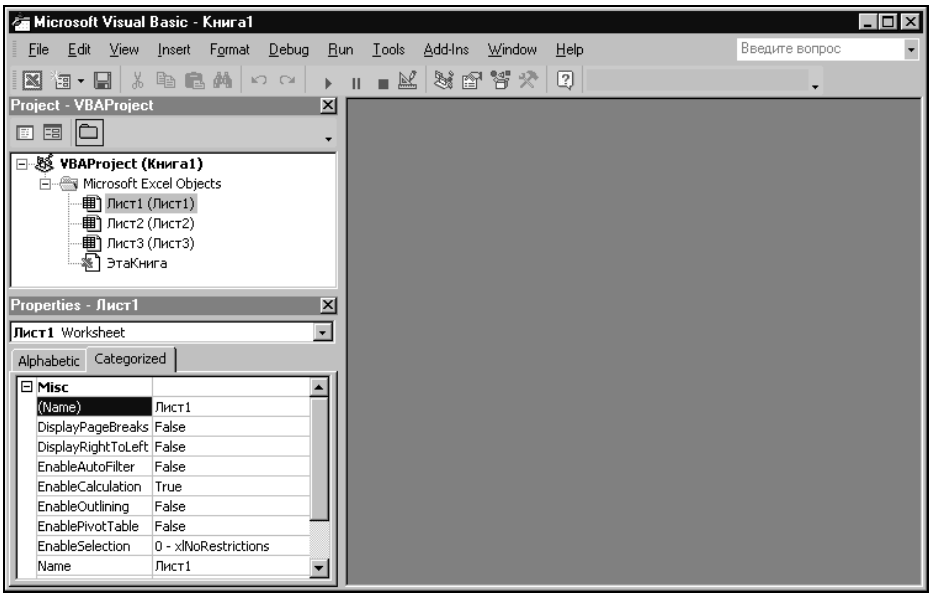


Рис. 1.1. Редактор Visual Basic

Она имеет стандартный вид для Windows-приложений: строка меню, панель инструментов (в данном случае **Standard**) и два окна **Project — VBAProject** и **Properties**. Пока нас интересует только окно **Project — VBAProject**. В нем отображается реестр модулей и форм, входящих в создаваемый проект. Модуль, упрощенно говоря, это лист (не путать с рабочим листом), в котором набирается код. Двойным щелчком на значке модуля в окне **Project — VBAProject** можно открыть соответствующий модуль. Значок активного модуля в окне **Project — VBAProject** выделяется серым цветом. В VBA у каждого рабочего листа имеется по модулю. Кроме того, рабочая книга обладает своим, и если в проекте создаются пользовательские формы, то каждая из них также имеет по модулю. В проект тоже можно добавлять модули классов для описания создаваемых пользовательских классов, но в данный момент ни один из этих модулей нас не интересует. Единственный модуль, который нам нужен, — это стандартный модуль. Для того чтобы его добавить в проект, выберите команду **Insert | Module**.

Структура кода функции пользователя

Итак, стандартный модуль создан, можно перейти к написанию функции пользователя. В общем случае, функция пользователя имеет следующий вид:

```
Function ИмяФункции (СписокПараметров)
   Инструкции
End Function
```

- *СписокПараметров* — это список параметров, от которых зависит функция. Разделителем в списке параметров является запятая;
- *Инструкции* — это последовательность инструкций, выполняемых при нахождении значения функции. В совокупности они образуют так называемое тело функции. Важной особенностью функции пользователя является то, что носителем возвращаемого ее значения является *ИмяФункции*. Поэтому среди инструкций должен присутствовать по крайней мере один оператор, который присваивает имени функции значение какого-либо выражения.

Примечание

Допустим досрочный выход из функции по инструкции `Exit Function`. В теле функции может располагаться несколько инструкций `Exit Function`.

Ваша первая функция пользователя

Итак, теперь можно перейти к первому примеру построения функции пользователя. Давайте для начала построим очень простенькую функцию, скажем $F(x) = x^2$.

Для того чтобы это сделать, достаточно в стандартном модуле (рис. 1.2) набрать следующий код (листинг 1.1).

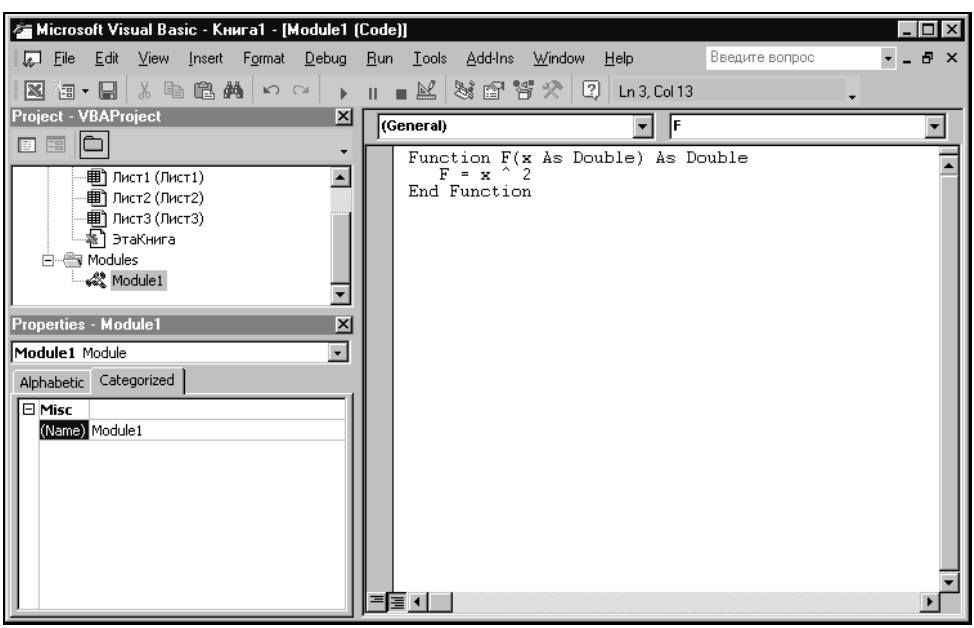


Рис. 1.2. Код пользовательской функции в модуле

Листинг 1.1. Пользовательская функция

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

В VBA имеется универсальный тип данных `Variant`, который подразумевается по умолчанию, если явно не был объявлен тип переменной или функции. Поэтому ту же самую функцию можно было бы закодировать следующим образом (листинг 1.2).

Листинг 1.2. Пользовательская функция с использованием типа `Variant`

```
Function F(x)
   F = x ^ 2
End Function
```

Примечание

Еще раз обратите внимание, что код пользовательской функции вводится в стандартном модуле, который добавляется в проект командой **Insert | Module**. В проекте много модулей, случайно не перепутайте. Активный модуль в окне **Project — VBAProject** выделяется серым цветом.

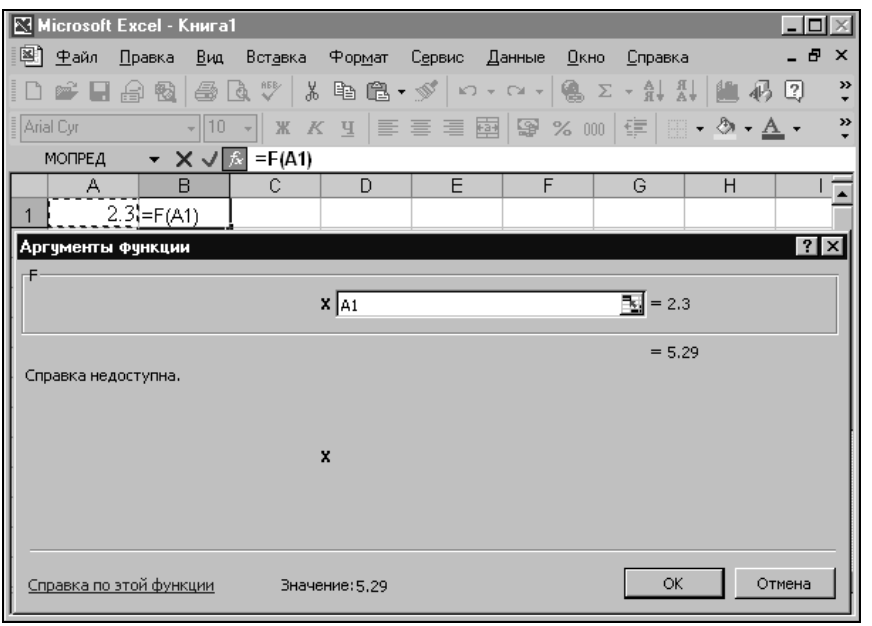


Рис. 1.3. Вычисление значения пользовательской функции при помощи мастера функций

Вот, собственно говоря, и все, пользовательская функция создана. По умолчанию она попадает в раздел **Определенные пользователем** списка **Категория** окна **Мастер функций**. Найдем, например, значение этой функции при $x = 2{,}3$. Для этого:

1. Введите число `2.3` в ячейку **A1** рабочего листа.
2. Выберите ячейку **B1**, в которой найдем значение функции.
3. Выберите команду **Вставка | Функция**.
4. В первом окне мастера функций выберите функцию `F` и нажмите кнопку **OK**.
5. Во втором окне мастера функций в поле **X** введите ссылку на ячейку **A1** и нажмите кнопку **OK** (рис. 1.3).

Значение функции найдено.

Вычисление стоимости партии продаваемых книг при помощи пользовательской функции

Вы можете сказать, что функция из последнего раздела очень простая, и не было большого смысла городить столько ради ее нахождения. В этом смысле, конечно, вы будете полностью правы. Цель предыдущего примера была совершенно в другом — не облегчить вам жизнь при помощи пользовательской функции, а просто показать, как они строятся и используются.

Рассмотрим более сложный пример построения функции пользователя. На минуту представим себе, что вы менеджер издательства по оптовой продаже книг. Для привлечения покупателей в вашем издательстве введена прогрессивная шкала цен. Если продается от 100 до 200 экземпляров книги, то скидка от ее отпускной цены составляет 7%, если продается от 201 до 300 экземпляров, то скидка составляет 10%, а если свыше 300 экземпляров, то — 15%. Кроме того, для постоянных клиентов предусмотрена дополнительная скидка в размере 5%. Создадим функцию пользователя с именем `Стоимость` для расчета стоимости партии книг. Параметры этой функции назовем `ЦенаОднойКниги`, `Количество` и `Скидка`. Для параметра `Скидка` предусмотрим только два допустимых значения: 1 — для постоянных клиентов и 0 — для всех остальных. Определим пользовательскую функцию `Стоимость` следующим кодом (листинг 1.3).

Листинг 1.3. Расчет стоимости партии книг

```
Function Стоимость(ЦенаОднойКниги, Количество, Скидка)
   If Количество < 100 Then
      СтоимостьБезСкидки = ЦенаОднойКниги * Количество
```

```
   ElseIf Количество <= 200 Then
      СтоимостьБезСкидки = ЦенаОднойКниги * Количество * 0.93
   ElseIf Количество <= 300 Then
      СтоимостьБезСкидки = ЦенаОднойКниги * Количество * 0.9
   Else
      СтоимостьБезСкидки = ЦенаОднойКниги * Количество * 0.85
   End If

   If Скидка = 0 Then
      Стоимость = СтоимостьБезСкидки
   Else
      Стоимость = СтоимостьБезСкидки * 0.95
   End If
End Function
```

Итак, функция пользователя `Стоимость` создана. Благодаря тому, что в VBA допустимо применение русскоязычных имен, текст написанной программы ясен и прозрачен для понимания. Кроме того, это обеспечивает и простоту использования диалогового окна мастера функций для данной функции (рис. 1.4).

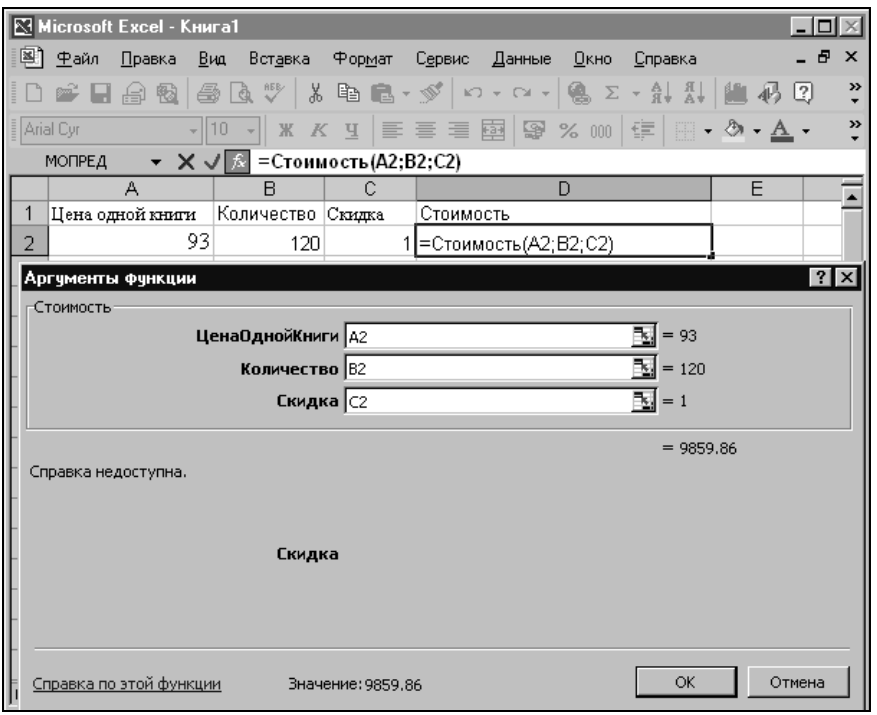


Рис. 1.4. Расчет стоимости партии книг

Названия всех параметров функции Стоимость в окне мастера функций выводятся также на русском. Ясная структура диалогового окна позволяет применять функцию Стоимость любому пользователю, даже не владеющему VBA. Для ее использования достаточно только знать имя этой функции.

Параметрами пользовательских функций могут быть ссылки на диапазоны

Пример из предыдущего раздела о вычислении стоимости партии книг звучит уже убедительно. VBA действительно может облегчить жизнь пользователя. Возникает только один вопрос по двум приведенным примерам. В созданных в них пользовательских функциях в качестве значений параметров были только ссылки на ячейки. Можно ли построить пользовательскую функцию, у которой в качестве значений параметров могут быть ссылки на диапазоны ячеек? Скажем, возможно ли построить некоторый аналог функции рабочего листа СУММ, которая возвращает сумму значений из указанных диапазонов ячеек. Ответ на этот вопрос положителен. И задачу решает следующий код (листинг 1.4).

Листинг 1.4. Нахождение суммы

```
Function MySum(ByVal rng As Range) As Double
   Dim c As Range
   Dim s As Double
   s = 0
   For Each c In rng.Cells
      s = s + c.Value
   Next
   MySum = s
End Function
```

Автоматизация работы рабочего листа при помощи элементов управления

При помощи панели инструментов **Элементы управления**, которая отображается на экране выбором команды **Вид | Панели инструментов | Элементы управления**, на рабочем листе можно размещать различные элементы управления: кнопку, список, переключатель и т. д (рис. 1.5). Подробно об элементах управления мы будем говорить в *главе 6*.

В модуле рабочего листа можно создать процедуры, обрабатывающие те или иные события, происходящие с элементами управления. Например, нажатие

кнопки, выбор элемента из списка, выбор переключателя, установка флажка и т. д., которые будут автоматически приводить к тем или иным вычислением, построению диаграмм или смене их типа и т. д. В последующих разделах будут приведены простейшие примеры подобной автоматизации работы с рабочим листом. Первоначально ограничимся только одним элементом управления — **Кнопка**, создаваемым кнопкой панели инструментов **Элементы управления**, и одним событием, связанным с кнопкой, — `Click`, которое генерируется при нажатии на кнопку.

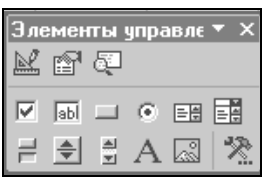


Рис. 1.5. Панель инструментов для рабочего листа

Работа с макрорекордером

Написание программы, это, в каком-то смысле, обучение компьютера делать то, что вам нужно от него. Для быстрого получения чернового варианта кода, используемого в создаваемом приложении, отлично подходит макрорекордер. *Макрорекордер* — это транслятор, создающий программу (макрос) на языке VBA, которая является результатом перевода на языке VBA действий пользователя с момента запуска макрорекордера до окончания записи макроса.

Давайте, для примера, посмотрим, какой записывается в макрос код при активизации рабочего листа. Затем, в следующем разделе, мы используем его для построения простейшего автоматизированного содержания рабочей книги.

Итак, пусть у нас активизирован рабочий лист **Лист1**, и мы запишем макрос, который получается при активизации рабочего листа **Лист2**.

Для активизации макрорекордера выберите команду **Сервис | Макрос | Начать запись**. Появится диалоговое окно **Запись макроса** (рис. 1.6). Это диалоговое окно позволяет задать параметры макроса.

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

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

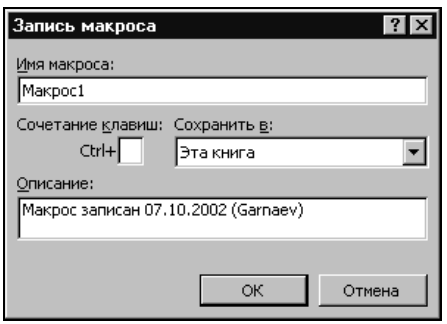


Рис. 1.6. Диалоговое окно **Запись макроса**

Итак, в диалоговом окне **Запись макроса** в поле **Имя макроса** введем `АктивизироватьЛист`, а в поле **Описание** — `Активизация рабочего листа Лист2`, и нажмем кнопку **OK**. Появится плавающая панель инструментов с кнопкой **Остановить запись** ■. Теперь все производимые действия будут записываться до тех пор, пока не будет нажата эта кнопка или не будет выбрана команда **Сервис | Макрос | Остановить запись**.

Далее щелчком на ярлыке рабочего листа **Лист2** активизируйте его и остановите запись макроса выбором соответствующей команды.

Для просмотра только что записанной процедуры выберите команду **Сервис | Макрос | Макросы**, что приведет к отображению диалогового окна **Макрос** (рис. 1.7).

В диалоговом окне **Макрос** в списке выделим макрос и нажмем кнопку **Изменить**. На экране отобразится окно редактора VBA с активизированным

стандартным модулем, в котором будет код (листинг 1.5) только что записанного макроса.

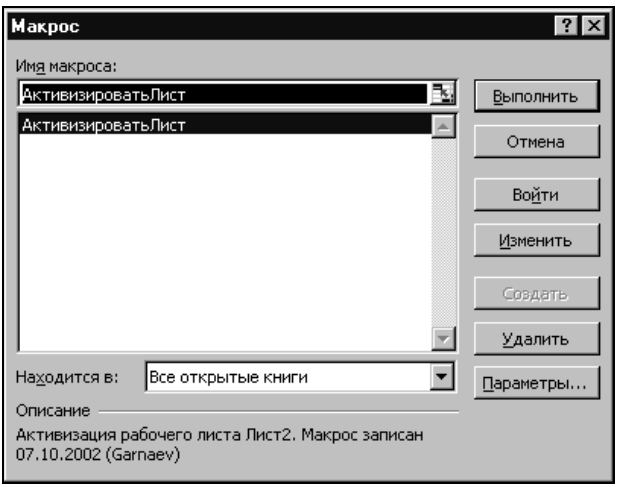


Рис. 1.7. Диалоговое окно **Макрос**

Листинг 1.5. Макрос, активизирующий рабочий лист Лист2. Стандартный модуль

```
Sub АктивизироватьЛист()
'
' АктивизироватьЛист Макрос
' Активизация рабочего листа Лист2. Макрос записан 07.10.2002 (Garnaev)
'

'
    Sheets("Лист2").Select
End Sub
```

Структура кода процедуры

Итак, макрос записан, но прежде чем идти дальше, нам необходимо пополнить теоретический багаж. Отметим, что макрорекордер на самом деле в модуле создал процедуру. В общем случае процедура имеет следующий вид:

```
Sub ИмяПроцедуры(СписокПараметров)
   Инструкции
End Sub
```

- *СписокПараметров* — это список параметров, от которых зависит функция. Разделителем в списке параметров является запятая. Список параметров, вообще говоря, может быть и пуст, как это и случилось в листинге 1.5;
- *Инструкции* — это последовательность инструкций, выполняемых при выполнении процедуры. В совокупности они образуют так называемое тело процедуры.

Примечание

Допустим досрочный выход из процедуры по инструкции `Exit Sub`. В теле процедуры может располагаться несколько инструкций `Exit Sub`.

Процедура обработки события

Кроме обычных процедур, в VBA имеются процедуры, обрабатывающие события, связанные с тем или иным объектом. В общем случае такие процедуры имеют следующий вид:

```
[Private] Sub ИмяОбъекта_ИмяСобытия(СписокПараметров)
   Инструкции
End Sub
```

Ключевое слово `Private` является модификатором доступа и обозначает, что процедура видна другим процедурам только в модуле, в котором она располагается. Например, процедура обработки события активизации рабочего листа имеет вид:

```
Private Sub Worksheet_Activate()
   Инструкции
End Sub
```

Элемент управления *Кнопка*, создание процедуры обработки события и автоматическое перелистывание рабочих листов

В данном разделе создадим на рабочем листе **Лист1** кнопку, нажатие на которую будет приводить к активизации рабочего листа **Лист2**. Первоначально выберите рабочий лист **Лист1**.

Прежде чем переходить к осуществлению наших планов, поговорим о том, как создается кнопка на рабочем листе, что собой представляет элемент управления **Кнопка**, и как пишется код обработки события "нажатие на кнопку".

Прежде всего переведите MS Excel в режим конструктора элементов управления. Для этого нажмите кнопку **Режим конструктора** панели инструментов **Элементы управления**.

Теперь можно нажать кнопку и нарисовать элемент управления **Кнопка** точно так же, как и при помощи любого графического редактора, в том месте рабочего листа, где вы задумали расположить кнопку.

Примечание

Как и любой графический объект, кнопку можно рисовать при нажатой клавише <Shift>, что обеспечит ей квадратную форму, либо при нажатой клавише <Alt> — для привязки кнопки к сетке рабочего листа. У созданной кнопки при помощи маркеров изменения размеров можно задать размеры, а при помощи маркера перемещения — установить ее местоположение.

На поверхности первого созданного элемента управления **Кнопка** автоматически будет отображена надпись **CommandButton1** (рис. 1.8). Если вы создадите второй элемент управления **Кнопка**, то на его поверхности отобразится надпись **CommandButton2** и т. д.

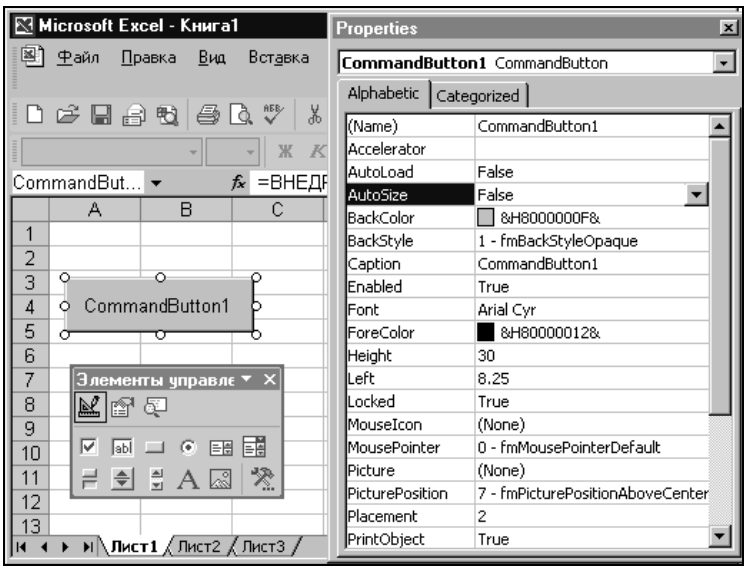


Рис. 1.8. Кнопка и окно **Properties**

Элемент управления **Кнопка** является объектом, т. е. он, как и любой объект, обладает свойствами, методами и событиями. Надпись, отображаемая на поверхности элемента управления **Кнопка**, задается значением свойства `Caption`. Кроме того, элемент управления **Кнопка** имеет свойство `Name`, которое в коде идентифицирует его как объект. В данном случае, оно тоже равно `CommandButton1`. Изменим, значение свойства `Caption`, т. е. надпись, ото-

бражаемую на поверхности кнопки с **CommandButton1** на **Лист2** (с тем, чтобы подчеркнуть, что эта кнопка будет активизировать рабочий лист **Лист2**). Для этого нажмите кнопку **Свойства**  панели инструментов **Элементы управления**. На экране отобразится окно **Properties**. Введите в поле `Caption` этого окна значение `Лист2` и закройте окно **Properties**.

Примечание

Сразу же стоить отметить, что в Excel существуют две похожие панели инструментов — **Элементы управления** и **Формы**. Панель инструментов **Формы** унаследована от версии Excel 5.0 и в основном используется для совместимости приложений, созданных в старых версиях.

Теперь можно перейти к созданию кода процедуры, обрабатывающей событие "нажатие на кнопку". В результате обработки этого события должен активизироваться рабочий лист **Лист2**. Для этого дважды щелкните на созданной кнопке (напоминаем, что все это еще происходит в режиме конструктора). В результате откроется редактор VBA с активизированным модулем рабочего листа (в данном случае, **Лист1**). Кроме того, двойной щелчок на кнопке создаст в модуле первую и последнюю инструкции процедуры обработки события "нажатие на кнопку" (листинг 1.6, *а*).

Листинг 1.6, *а*. Первая и последняя инструкции процедуры обработки события "нажатие на кнопку". Модуль рабочего листа Лист1

```
Private Sub CommandButton1_Click()

End Sub
```

Вот теперь нам пригодится ранее созданный макрос (см. листинг 1.5). Из его кода видно, что активизацией (в данном случае, точнее сказать, выбором) рабочего листа управляет одна инструкция:

```
Sheets("Лист2").Select
```

Через буфер обмена скопируйте эту инструкцию из макроса в процедуру обработки события "нажатие на кнопку" (листинг 1.6, *б*). Конечно, ее можно было бы набрать и вручную. Но это довольно медленно и чревато появлением случайных опечаток, неизбежно время от времени тут и там возникающих при наборе кода.

Листинг 1.6, *б*. Процедура обработки события "нажатие на кнопку", при котором будет активизирован рабочий лист Лист2. Модуль рабочего листа Лист1

```
Private Sub CommandButton1_Click()
   Sheets("Лист2").Select
End Sub
```

Итак, процедура создана. Осталось только вернуться на рабочий лист **Лист1**. Созданная кнопка будет обрабатывать событие (нажатие на нее) только после выхода из режима конструктора. Поэтому отключите режим конструктора нажатием кнопки **Режим конструктора** панели инструментов **Элементы управления**. Теперь можно и протестировать созданную кнопку. Нажмите на нее, и если вы все правильно сделали, то это приведет к активизации рабочего листа **Лист2**.

В качестве упражнения и закрепления только что прочитанного материала создайте на рабочем листе **Лист1** еще одну кнопку, которая будет активизировать рабочий лист **Лист2**.

Построение шаблона таблицы

Предположим, что вы менеджер фирмы, например ООО "Бескрайние просторы", и каждый месяц составляете таблицу учета расходов (рис. 1.9). Давайте попытаемся облегчить вашу работу — автоматизировать процесс создания шаблона таблицы, в которую останется только ввести данные.

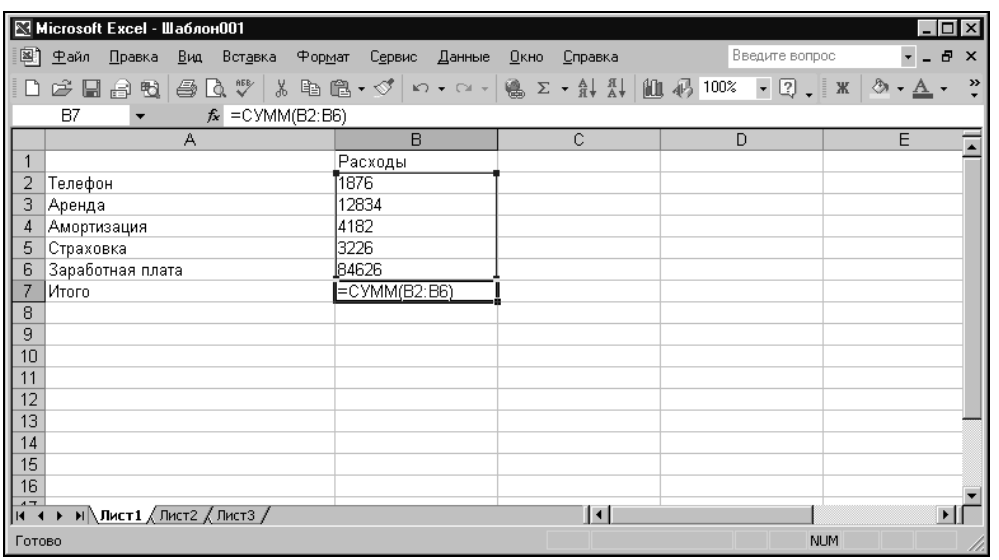


Рис. 1.9. Шаблон таблицы расходов

Как всегда, при написании кода поможет макрорекордер.

1. Запустите макрорекордер на выполнение выбором команды **Сервис | Макрос | Начать запись**.
2. Выберите ячейку **B1** и введите в нее `Расходы`.

3. Выберите ячейку **A2** и введите в нее `Телефон`.
4. Выберите ячейку **A3** и введите в нее `Аренда`.
5. Выберите ячейку **A4** и введите в нее `Амортизация`.
6. Выберите ячейку **A5** и введите в нее `Страховка`.
7. Выберите ячейку **A6** и введите в нее `Заработная плата`.
8. Выберите ячейку **A7** и введите в нее `Итого`.
9. Выберите ячейку **B7** и введите в нее формулу `=СУММ(B2:B6)`.
10. Выберите столбец **A** и команду **Формат | Столбец | Автоподбор ширины**.
11. Завершите работу макрорекордера выбором команды **Сервис | Макрос | Остановить запись**.

В результате на листе стандартного модуля будет записан следующий макрос (листинг 1.7).

Листинг 1.7. Макрос по составлению шаблона отчета

```
Sub Макрос1()
'
' Макрос1 Макрос
' Макрос записан 07.10.2002 (Garnaev)
'

'
    Range("B1").Select
    ActiveCell.FormulaR1C1 = "Расходы"
    Range("A2").Select
    ActiveCell.FormulaR1C1 = "Телефон"
    Range("A3").Select
    ActiveCell.FormulaR1C1 = "Аренда"
    Range("A4").Select
    ActiveCell.FormulaR1C1 = "Амортизация"
    Range("A5").Select
    ActiveCell.FormulaR1C1 = "Страховка"
    Range("A6").Select
    ActiveCell.FormulaR1C1 = "Заработная плата"
    Range("A7").Select
    ActiveCell.FormulaR1C1 = "Итого"
    Range("B7").Select
    ActiveCell.FormulaR1C1 = "=SUM(R[-5]C:R[-1]C)"
    Columns("A:A").Select
    Selection.Columns.AutoFit
End Sub
```

Давайте обсудим, как из этого непропорционально длинного по сравнению с проделанной работой и неказистого макроса составить разумный код для создания шаблона отчета.

Первые четырнадцать строк кода макроса осуществляют попарно ввод текстовой информации в выбранные ячейки рабочего листа. Имеет смысл каждую из этих пар инструкций заменить одной. Например, вместо

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

можно ограничиться только инструкцией

```
Range("B1").Value= "Расходы"
```

Следующие две инструкции реализуют ввод формулы `=СУММ(B2:B6)` в ячейку **B7**. При этом в макросе эта формула записана в относительном формате **R1C1**, что выглядит довольно не к месту:

```
Range("B7").Select
ActiveCell.FormulaR1C1 = "=SUM(R[-5]C:R[-1]C)"
```

Разумно эти две инструкции заменить только одной, используя формат **A1** и формулу локальной версии.

```
Range("B7").FormulaLocal = "=СУММ(B2:B6)"
```

Последние две инструкции макроса реализуют автоматический подбор ширины столбца **A** так, чтобы в нем помещались все введенные в него строки текста.

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

Разумно сократить код, заменив эти две инструкции одной:

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

Кроме того, для пользователя было бы удобно, чтобы имя рабочего листа совпадало с названием месяца, за который составляется отчет. Поэтому естественно в процедуре добавить инструкции, которые привели бы к отображению на экране диалогового окна, предлагающего изменить название листа. Если пользователь соглашается с этим предложением, то он вводит в поле ввода появившегося диалогового окна новое имя рабочего листа и нажимает на кнопку **OK**, а процедура сама уж переименует лист.

Итак, код окончательного варианта процедуры составления шаблона прозрачен, он состоит всего из нескольких инструкций, и, что важно, значительно более функционален, чем макрос, на основе которого он был составлен (листинг 1.8).

Листинг 1.8. Процедура составления шаблона отчета

```
Sub ChartBuilder()
   Dim sheetName As String
   sheetName = InputBox("Введите имя листа")
   If sheetName <> Empty Then
      ActiveSheet.Name = sheetName
   End If
   Range("B1").Value = "Расходы"
   Range("A2").Value = "Телефон"
   Range("A3").Value = "Аренда"
   Range("A4").Value = "Амортизация"
   Range("A5").Value = "Страховка"
   Range("A6").Value = "Заработная плата"
   Range("A7").Value = "Итого"
   Range("B7").FormulaLocal = "=СУММ(B2:B6)"
   Columns("A:A").AutoFit
End Sub
```

Управление диаграммой

Предположим, что вы менеджер фирмы "Иван Дубленкин и К°", и вам надо создать годовой отчет о доходах фирмы, причем построить диаграммы дохода только за январь, за январь и декабрь, с января по март и т. д., всего 12 диаграмм. Конечно, можно построить все эти диаграммы, а можно обойтись и одной. В этом случае надо применить инструмент, позволяющий вводить заданный временной интервал, а диаграмма автоматически должна перестраиваться в соответствии с ним. Тут, конечно, на помощь приходят элементы управления — либо **Список**, либо **Поле со списком**. Итак:

1. В диапазон **A2:A13** введите названия месяцев (рис. 1.10).
2. В диапазон **B2:B13** введите доходы фирмы.
3. По этим двум диапазонам постройте диаграмму.
4. На рабочем листе расположите **Поле со списком**, которое и позволит произвести выбор временного интервала.
5. При помощи окна **Properties** установите его значение `ListFillRange` равным **A2:A13**. Это свойство заполняет список на основе данных из указанного диапазона.
6. В модуле рабочего листа наберите следующий код (листинг 1.9).

Вот и все! Проект готов.

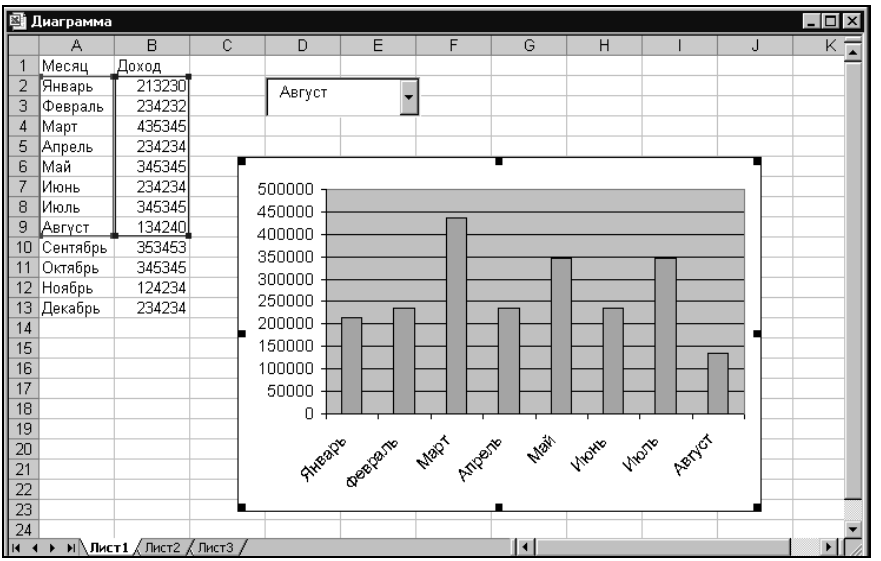


Рис. 1.10. Управление диаграммой

Листинг 1.9. Управление диаграммой

```
Private Sub ComboBox1_Change()
   Dim r As Integer
   ActiveSheet.ChartObjects(1).Activate
   r = ComboBox1.ListIndex + 2
   With ActiveChart
     .SetSourceData _
         Source:= _
           Sheets(1).Range(Cells(2, 2), Cells(r, 2)), PlotBy:=xlColumns
          .SeriesCollection(1).XValues _
              = Sheets(1).Range(Cells(2, 1), Cells(r, 1))
   End With
End Sub
```

Создание скриптов

Язык программирования VBScript, являющийся урезанной версией VBA, позволяет писать сценарии как для Web, так и Windows.

Создание скриптов для Windows

Windows-сценарии помогают пользователю решать множество рутинных задач типа создания ярлыка на рабочем столе, очистки папок и т. д. Файлы

Windows-сценариев являются текстовыми VBS-файлами. Например, создадим Windows-сценарий, который отображает окно с подсказкой дня. В любом текстовом редакторе (например, Notepad) наберите следующий код (листинг 1.10). Сохраните созданный код в файл с расширением vbs, например, Tips.vbs в корневом каталоге диска C:. Поздравляю, ваш первый Windows-сценарий готов. Осталось только протестировать его работоспособность. Для этого в любой программе просмотра (например, в Проводнике) просто щелкните на значке файла (рис. 1.11). Можно также на рабочем столе создать ярлык, которому назначить следующий объект, где Wscript.exe как раз и является той программой, которая проигрывает сценарии:

```
C:\WINDOWS\wscript.exe c:\tips.vbs
```

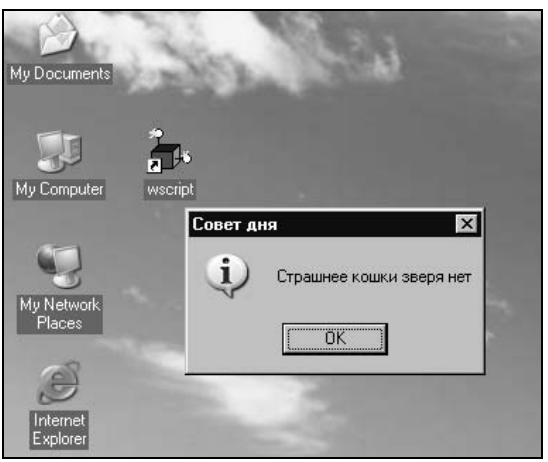


Рис. 1.11. Скрипт для Windows с советом дня

Листинг 1.10. Совет дня. Файл Tips.vbs

```
Dim d(2), n
d(0)="Страшнее кошки зверя нет"
d(1)="Последняя ошибка всегда бывает предпоследней"
d(2)="Если ничего другого не получается, " & vbCr & _
     "то внимательно прочитайте инструкции"
Randomize
n = CInt(2*Rnd())
Msgbox d(n), vbInformation, "Совет дня"
```

Создание скриптов для Web на стороне клиента

Используя упрощенную вариацию VBA, а именно VBScript, можно писать код непосредственно в HTML-файлах, и этот код будет проигрываться при

загрузке такого файла в браузер, т. е. на стороне клиента. Такой код должен быть помещен внутрь парного тега `<script>`, атрибут `language` которого специфицирует язык, на котором написан сценарий. В качестве примера приведем следующий код (листинг 1.11), который обеспечивает отображение окна с приветствием при щелчке на надписи "Hello, World!" (рис. 1.12).

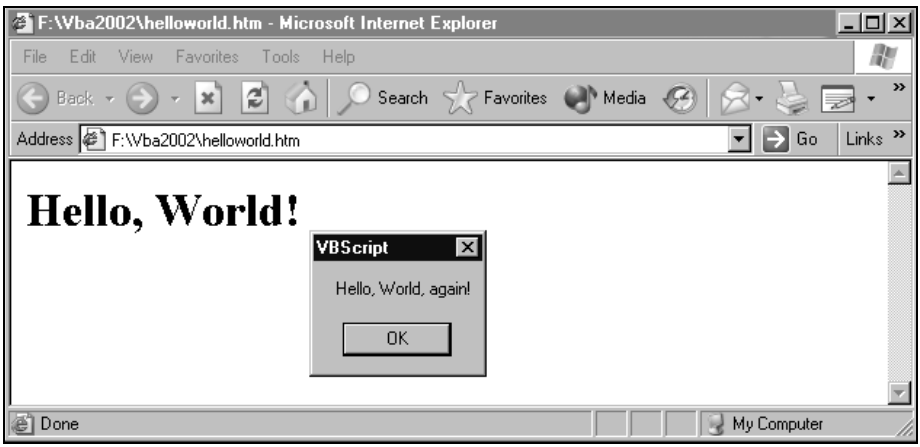


Рис. 1.12. Скрипт для Windows с отображением приветствия при щелчке на надписи

Листинг 1.11. Приветствие. Файл Hello.html

```
<html>
<head>
   <script language="VBScript">
      Sub hello()
         Msgbox "Hello, World, again!"
      End Sub
   </script>
</head>
<body>
<h1 onclick="hello">Hello, World!</h1>
</body>
</html>
```

Создание скриптов для Web на стороне сервера

Скрипты Microsoft Active Service Pages (ASP, активные серверные страницы), написанные на VBA (точнее, на его урезанной части VBScript) позволяют создавать сценарии, которые проигрываются на стороне сервера. Из всей выходной информации ASP-сценарии передают на компьютер клиента

только текст и HTML-теги, где они воспроизводятся в окне браузера. При помощи объектной модели ADO (ActiveX Data Object) ASP доступны базы данных, хранимые на сервере, а при помощи FSO — текстовые файлы, хранимые там же. Это позволяет легко создавать различные Web-приложения типа виртуальных каталогов. Если не быть чрезмерным буквоедом, то ADO можно описать как модель доступа к данным, оптимизированную для использования в Web-проектах. Для применения ASP необходимо установить Web-сервер Internet Information Server (IIS) корпорации Microsoft для работы с операционной системой Windows 2000 и Windows XP. Windows предоставляет возможность проигрывать серверные приложения на компьютере клиента. Для этого файл серверного приложения (например, Hello.asp, приведенный в листинге 1.12, *а*) расположите в каталоге C:\Inetpub\wwwroot компьютера клиента. Это корневой каталог сервера, создаваемый по умолчанию при его инсталляции. Клиенты, подключенные к вашему компьютеру с помощью HTTP-протокола, автоматически попадают в этот каталог. В каталоге имеется файл Default.htm, который открывается в браузере клиента при его соединении с сервером без указания имени документа. В поле **Адрес** браузера введите:

```
http://127.0.0.1/hello.asp
```

или

```
http://localhost/hello.asp
```

где `http://127.0.0.1` или `http://localhost` — зарезервированный IP-адрес для подключения к серверу, запущенном на компьютере, с которого поступил запрос на подключение.

В качестве примера создадим следующий код (листинг 1.12, *а*), который отображает в браузере различные приветствия в зависимости от времени загрузки документа (рис. 1.13). Для того чтобы компьютер мог различать, какая часть кода из ASP-файла должна выполняться на серверной стороне, надо воспользоваться тегом `<% statements %>`, где *statements* — операторы, выполняемые на сервере. Для определения времени загрузки документа применим функцию `Time`, которая возвращает искомую величину.

Листинг 1.12, *а*. Приветствие. Файл Hello.asp

```
<html>
<body>
<h1>Приветствие</h1>
   <% If Time>= #00:00:00# And Time < #12:00:00# Then %>
      <h2>Доброе утро!!!</h2>
   <% ElseIf Time>= #12:00:00# And Time < #18:00:00# Then %>
      <h2>Добрый день!!! </h2>
```

```
   <%Else%>
      <h2>Добрый вечер!!! </h2>
   <%End If %>
</body>
</html>
```

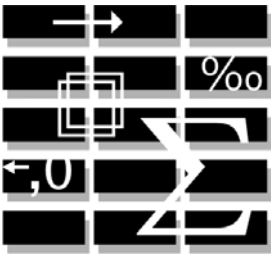


Рис. 1.13. Скрипт серверного сценария с приветствием, зависящим от времени загрузки документа в окно браузера

Если в браузере вы выберете команду **Вид | В виде HTML**, то сможете увидеть код, который получает клиент от сервера при проигрывании данного проекта. Например, загрузив файл Hello.asp в браузер в 13:00, можно получить следующий переданный код (листинг 1.12, *б*).

Листинг 1.12, *б*. Приветствие. Код, передаваемый клиенту

```
<html>
<body>
<h1>Приветствие</h1>
      <h2>Добрый день!!! </h2>
</body>
</html>
```



ГЛАВА 2

Интегрированная среда разработки

Где набирается код VBA

Код VBA набирается в редакторе Visual Basic. Для того чтобы попасть в этот редактор, выберите в MS Excel команду **Сервис | Макрос | Редактор Visual Basic**, или нажмите кнопку **Редактор Visual Basic** панели инструментов **Visual Basic**, или комбинацию клавиш <Alt>+<F11>. В результате вы попадете в интегрированную среду разработки приложений (IDE) редактора Visual Basic (рис. 2.1). Возвратиться из редактора Visual Basic в рабочую книгу можно нажатием кнопки **View Microsoft Excel** панели инструментов **Standard**. Среда разработки приложений редактора Visual Basic имеет стандартный вид для Windows-приложений: строка меню, панель инструментов (в данном случае **Standard**) и два окна **Project — VBAProject** и **Properties**.

Структура редактора VBA

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

Интерфейс редактора Visual Basic состоит из следующих основных компонентов:

- окна **Project — VBAProject** (Проект);
- окна редактирования кода;
- окна форм;
- окна свойств;
- панели инструментов.

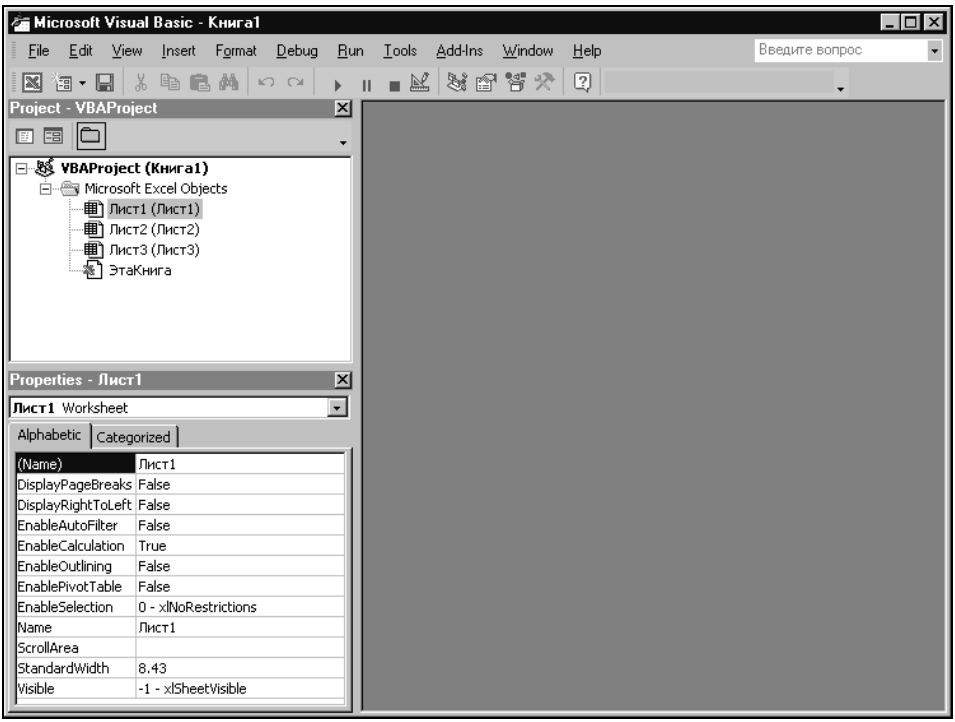


Рис. 2.1. Редактор Visual Basic

Окно *Project — VBAProject*

Окно **Project — VBAProject** в редакторе Visual Basic активизируется выбором команды **View | Project Explorer** или нажатием кнопки **Project Explorer** панели инструментов **Standard**. В окне **Project — VBAProject** представлена иерархическая структура файлов форм и модулей текущего проекта (рис. 2.2).

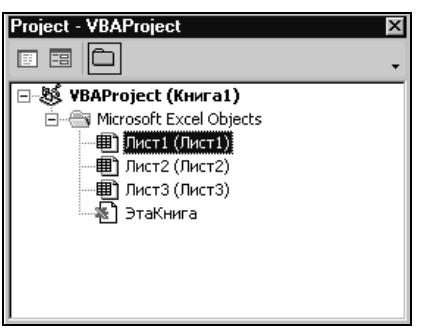


Рис. 2.2. Окно **Project — VBAProject**

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

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

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

Копирование модулей и форм из одного проекта в другой

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

Окно редактирования кода

Двойной щелчок на значке файла в окне проекта открывает окно редактора кода для соответствующего модуля (рис. 2.3). Открыть модуль в редакторе кода для соответствующего объекта (например, рабочего листа) можно также выбором значка этого объекта в окне проекта с последующим выбором команды **View | Code**. Вернуться назад из модуля к соответствующему объекту можно выбором команды **View | Object**.

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

View вкладки **Editor** диалогового окна **Options** (рис. 2.4), отображаемого на экране командой **Tools | Options**. Если установлен режим просмотра всех процедур модуля, то процедуры можно отображать с разделителями (горизонтальной чертой, разделяющей две соседние процедуры) или без них. Отображением или не отображением разделителей управляет флажок **Procedure Separator**.

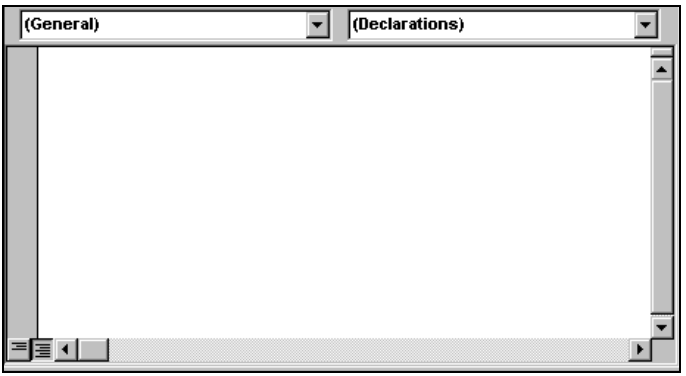


Рис. 2.3. Окно редактирования кода

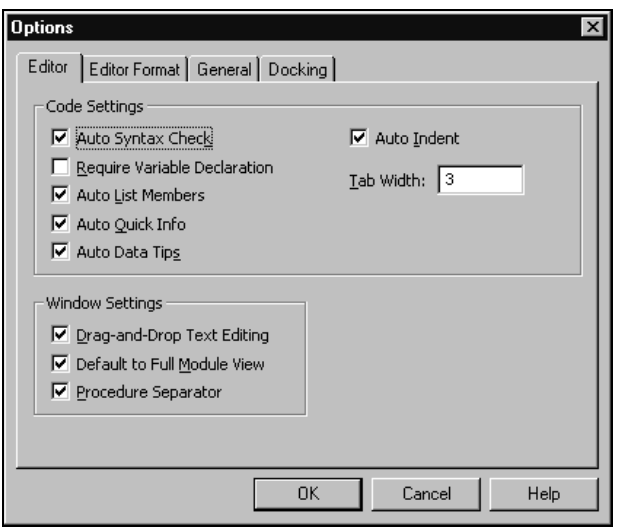


Рис. 2.4. Вкладка **Editor** диалогового окна **Options**

Два раскрывающихся списка в верхней части окна редактора кода облегчают ориентацию в процедурах. Левый раскрывающийся список позволяет выбрать объект, а правый — содержит список событий, допустимых для выбранного в левом списке объекта. Кроме того, выбор объекта и события

приводит к созданию в редакторе кода первой и последней инструкции процедуры обработки события, связанного с выбранным объектом, если такой еще не существовало.

Совет

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

Интеллектуальные возможности редактора кода

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

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

после ввода точки на экране отобразится список компонентов (рис. 2.5), которые логически завершают данную инструкцию. Двойной щелчок на выбранном элементе из этого списка или нажатие клавиши <Tab> вставляет выбранное имя в код программы. При этом использование клавиши <Tab> вместо мыши выглядит предпочтительней, т. к. эта клавиша находится прямо под рукой, и нажатие на нее производится только одним движением пальца левой руки, что не требует ни времени, ни усилий. В то же время работа с мышью, в данном случае, выглядит чрезмерно утомительным делом.

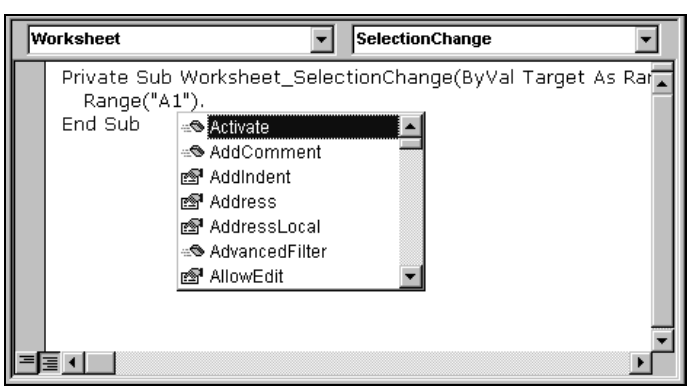


Рис. 2.5. Список компонентов

Автоматическое отображение списка компонентов происходит только при установленном флажке **Auto List Members** вкладки **Editor** диалогового окна **Options**, отображаемого на экране выбором команды **Tools | Options** (см. рис. 2.4).

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

Совет

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

Отображение списка компонентов, логически завершающих вводимую инструкцию, является одним из интеллектуальных качеств редактора кода. Этим качеством интеллектуальные ресурсы редактора кода не исчерпываются. Другим его интеллектуальным качеством является автоматическое отображение на экране сведений о процедурах, функциях, свойствах и методах после набора их имени (рис. 2.6).

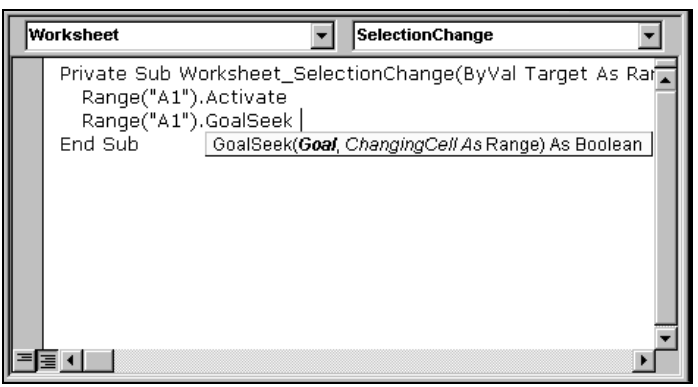


Рис. 2.6. Отображаемые сведения о вводимой процедуре

Автоматическое отображение на экране сведений о процедурах, функциях, свойствах и методах после ввода их имени происходит только при установленном флажке **Auto Data Tips** вкладки **Editor** диалогового окна **Options**, отображаемого на экране выбором команды **Tools | Options** (см. рис. 2.4).

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

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

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

Работа от примера является одним из краеугольных принципов миропознания и творчества. Стиль работы от примера очень ярко отображен в принципе чайника. Несмотря на всю его шутливость и кажущуюся абсурдность, в нем сформулирован многовековой опыт работы, который очень часто приводит к быстрым и эффективным результатам. Попытаемся описать принцип чайника следующим образом. У программиста спросили: "Как надо вскипятить воду, если на кухне есть спички, газовая плита, водопроводный кран и пустой чайник, стоящий на кухонном столе?" "Тривиально! — ответил программист. — Налейте в чайник воду, зажгите горелку газовой плиты и поставьте на нее чайник". Тогда программисту задают следующий вопрос: "На кухонном столе стоит чайник с водой, горелка газовой плиты зажжена. Как вскипятить чайник?" "Элементарно, Ватсон! — отвечает программист. — Надо просто вылить воду из чайника, погасить горелку, и мы оказываемся в условиях задачи, которая только что была успешно решена!"

Окно *UserForm* (Редактирование форм)

Для создания диалоговых окон разрабатываемых приложений в VBA используются формы. Редактор форм является одним из основных инструментов визуального программирования. Форма в проект добавляется выбором команды **Insert | UserForm**. В результате на экран выводится незаполненная форма с панелью инструментов **Toolbox** (рис. 2.7).

Используя панель инструментов **Toolbox** из незаполненной формы можно сконструировать любое требуемое для приложения диалоговое окно. Разме-

щение нового управляющего элемента на форме осуществляется следующей последовательностью действий:

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

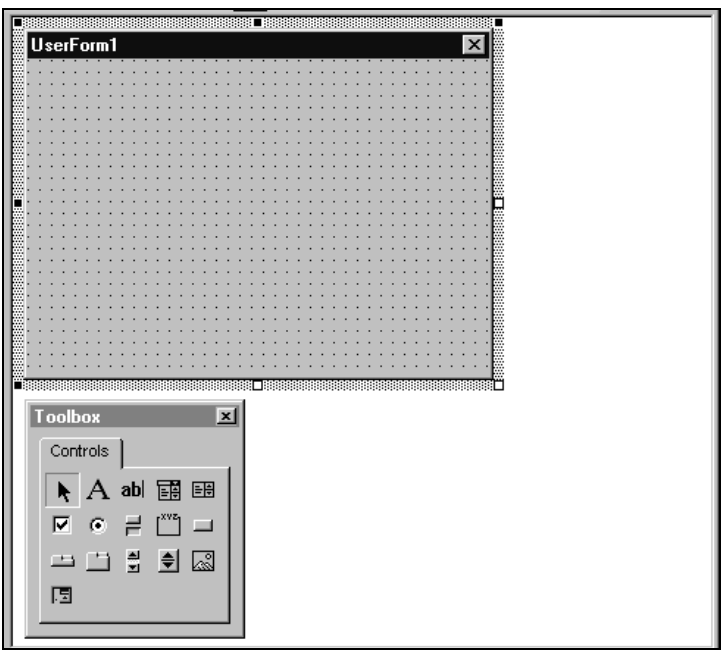


Рис. 2.7. Окно редактирования форм и панель инструментов **Toolbox**

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

Для облегчения размещения и выравнивания элементов управления используется сетка. Параметры сетки устанавливаются в группе **Form Grid Settings**

вкладки **General** диалогового окна **Options** вызываемого командой **Tools | Options**:

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

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

После формирования группы элементов управления легко совместно их перемещать и изменять размеры при помощи команд выпадающего меню **Format**, которые перечислены в табл. 2.1.

Таблица 2.1. *Команды выпадающего меню* ***Format***

Команда	Описание
Align | Lefts (Centers, Rights)	Выровнять группу элементов по левому краю (центру, правому краю)
Align | Tops (Middles, Bottoms)	Выровнять группу элементов по верхнему краю (середине, нижнему краю)
Align | To Grids	Выровнять по решетке
Make Same Size | Width (Height, Both)	Сделать элементы управления одной и той же ширины (высоты; и высоты, и ширины)
Size to Fit	Установить размеры элементов управления так, чтобы они совпадали с размерами отображаемых в них надписей, рисунков
Size to Grid	Установить размеры элементов управления по сетке
Horizontal Spacing | Make Equal (Increase, Decrease, Remove)	Установить равное по горизонтали расстояние между элементами управления (увеличить его, уменьшить, удалить)
Vertical Spacing | Make Equal (Increase, Decrease, Remove)	Установить равное по вертикали расстояние между элементами управления (увеличить его, уменьшить, удалить)

Таблица 2.1 *(окончание)*

Команда	Описание
Order | Bring to Front (**Sent to Back**)	Расположить элементы управления на переднем (заднем) плане

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

Окно *Properties* (Свойства)

В окне свойств перечисляются основные установки свойств выбранной формы или элемента управления. Используя это окно можно просматривать свойства и изменять их установки. Для просмотра свойств выбранного объекта надо либо щелкнуть кнопку **Properties Window** , либо выбрать команду **View | Properties Window**, либо нажать клавишу <F4> (рис. 2.8).

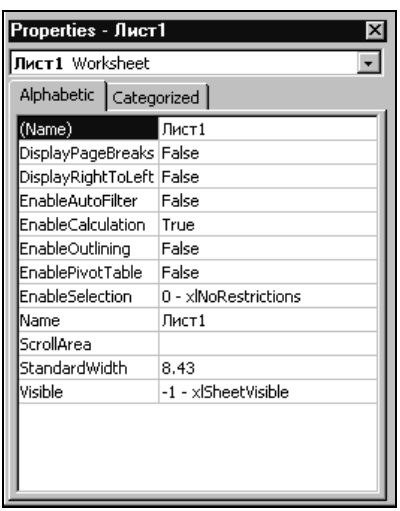


Рис. 2.8. Окно **Properties**

Окно свойств состоит из двух составных частей: верхней и рабочей. В верхней части окна свойств располагается раскрывающийся список, из которого можно выбрать любой элемент управления текущей формы или саму форму. Рабочая часть окна свойств состоит из двух вкладок: **Alphabetic** и **Categorized**, отображающих набор свойств в алфавитном порядке или по категориям. На обеих вкладках свойство `Name` (имя элемента управления) идет первым.

Изменяются значения свойств одним из следующих способов:

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

Совет

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

```
Private Sub cmdExit_Click()

End Sub
```

значительно более информативен, чем код:

```
Private Sub CommandButton1_Click()

End Sub
```

Окно *Object Browser* (Просмотр объектов)

Окно **Object Browser** отображается на экране выбором команды **View | Object Browser**, или нажатием кнопки **Object Browser** , или нажатием клавиши <F2> (рис. 2.9). В этом окне приведен список всех объектов, которые имеются в системе и которые можно использовать при создании проекта.

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

- раскрывающегося списка **Project/Library** в левом верхнем углу окна. В этом раскрывающемся списке можно выбрать различные проекты и библиотеки объектов. В частности, библиотеки объектов Excel, VBA, Office и VBAProject (объекты пользовательского проекта). Выбор в списке элемента **<All Libraries>** отображает список объектов всех библиотек;
- списка **Classes**. После выбора из раскрывающегося списка **Project/Library** просматриваемой библиотеки, например VBA, все классы объектов выбранной библиотеки выводятся в списке **Classes**;
- списка **Members**. После выбора класса из списка **Classes** просматриваемой библиотеки, например `FileSystem`, все компоненты выбранного класса выводятся в списке **Members**. При выделении строки в списке **Members** в нижней части окна **Object Browser** приводится дополнительная информация о выбранном компоненте. Кроме того, если нажать на

кнопку **Help**, расположенную на панели инструментов в правой верхней части окна **Object Browser**, то на экране отобразится окно **Справочник Visual Basic** с подробной справкой по выделенному компоненту.

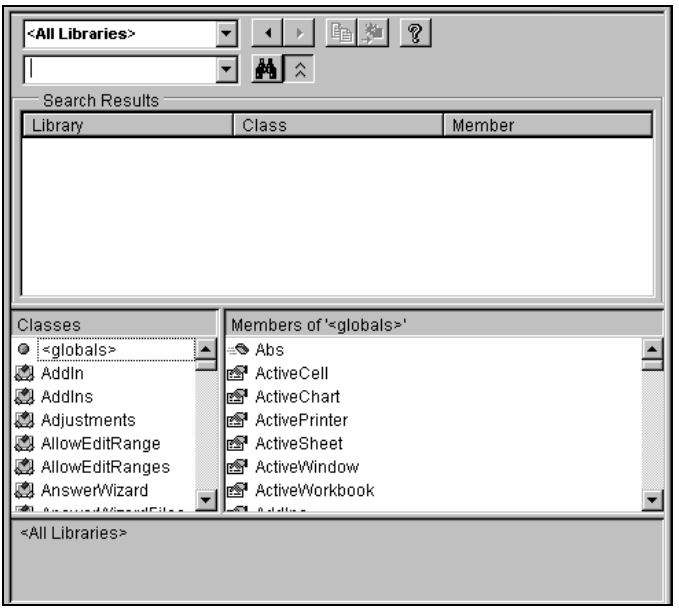
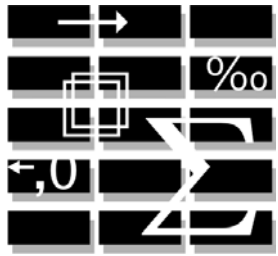


Рис. 2.9. Окно **Object Browser**



ГЛАВА 3

Базовые объекты

В данной главе рассматриваются базовые элементы объектной модели MS Excel: `Application`, `Workbook`, `Worksheet`, `Range`, `Window`, инкапсулирующие в себе данные о самом приложении, рабочей книге, диапазоне и окне. Кроме того, обсуждаются дочерние по отношению к данным объекты типа `Border` (граница), `Hyperlink` (гиперссылка), `Interior` (фон диапазона), `Font` (шрифт), `Protection` (данные об установленной защите), `Debug` (окно отладки).

Объектная модель

Объектная модель MS Excel представляет собой иерархию объектов, подчиненных одному объекту `Application`, который соответствует самому приложению MS Excel. Многие из этих объектов собраны в библиотеке объектов MS Excel, но некоторые из них, например объект `Assistant`, входят в библиотеку объектов MS Office, которая является общей для всех офисных приложений. На рис. 3.1—3.5 собраны основные компоненты объектной модели MS Excel. На этих рисунках имена единичных объектов находятся в прямоугольниках с более темным фоном. Имена объектов из семейств помещены в скобки после имени семейства. Объекты, справа от которых находятся треугольные стрелки, дополнительно раскрыты на других схемах. Каждый из объектов будет обсужден по ходу рассмотрения в книге, поэтому в данный момент ограничимся только представлением общей схемы объектной модели.

Полная и неявная ссылка на объект

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

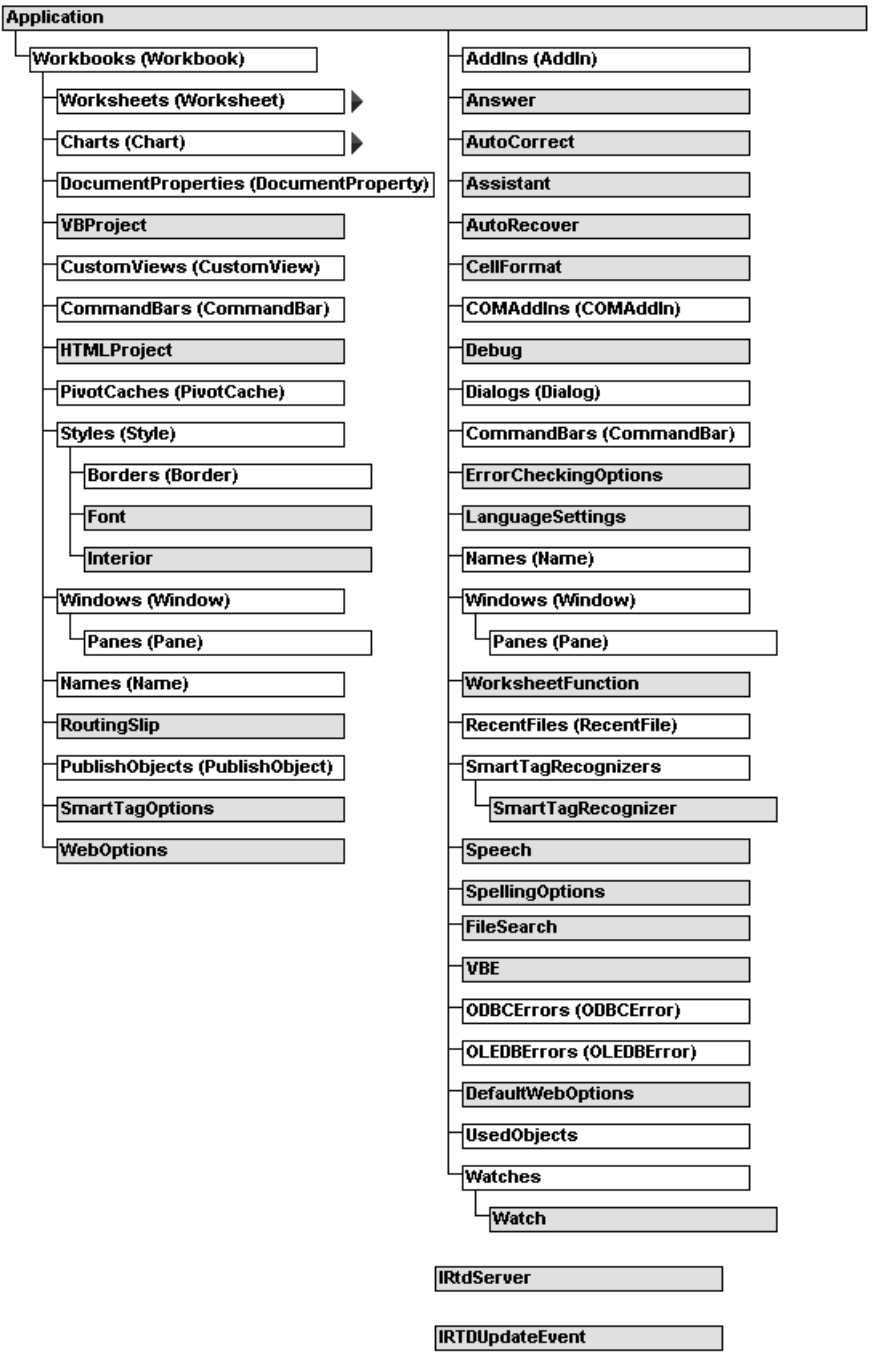


Рис. 3.1. Объект `Application` и непосредственно подчиняющиеся ему объекты

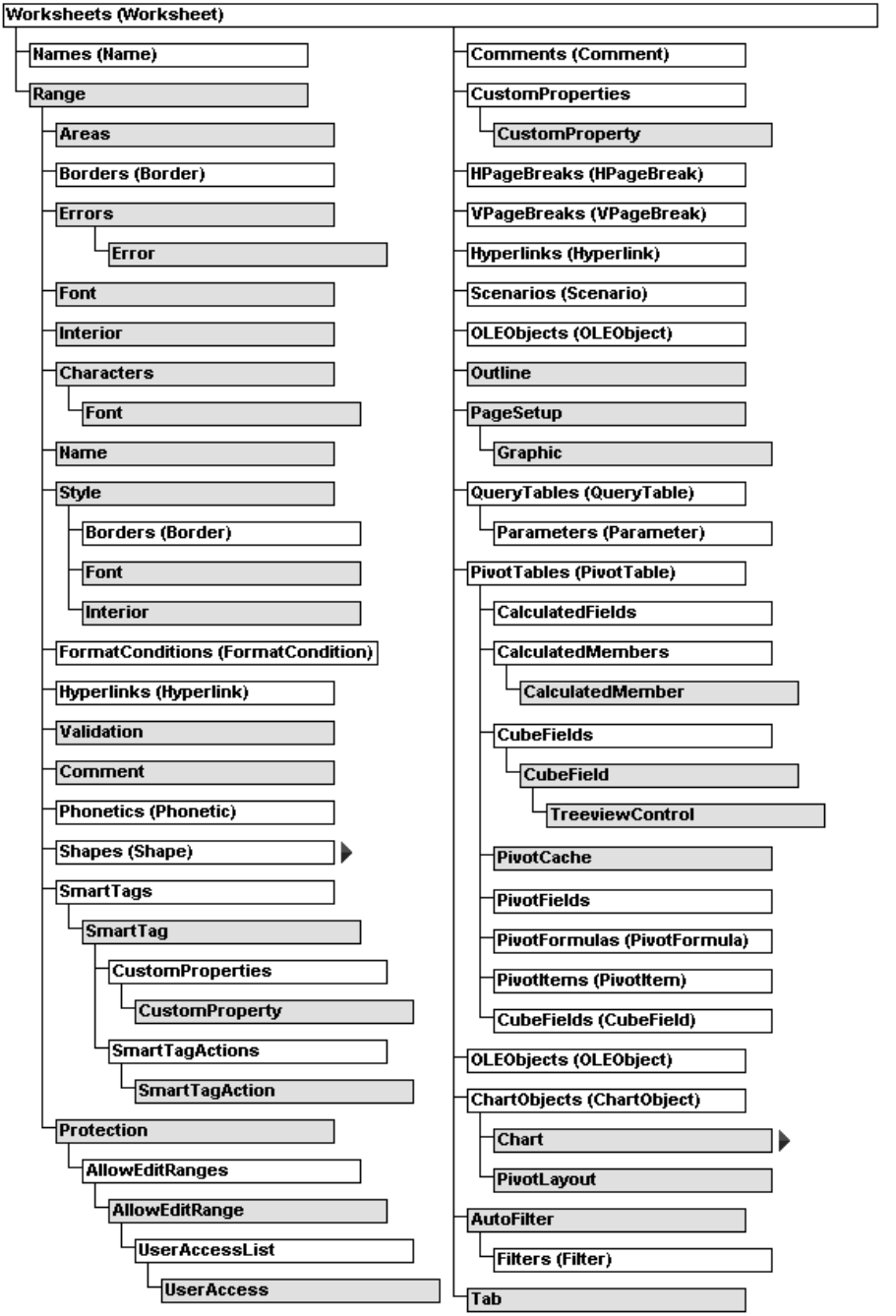


Рис. 3.2. Объект `Worksheet` и непосредственно подчиняющиеся ему объекты

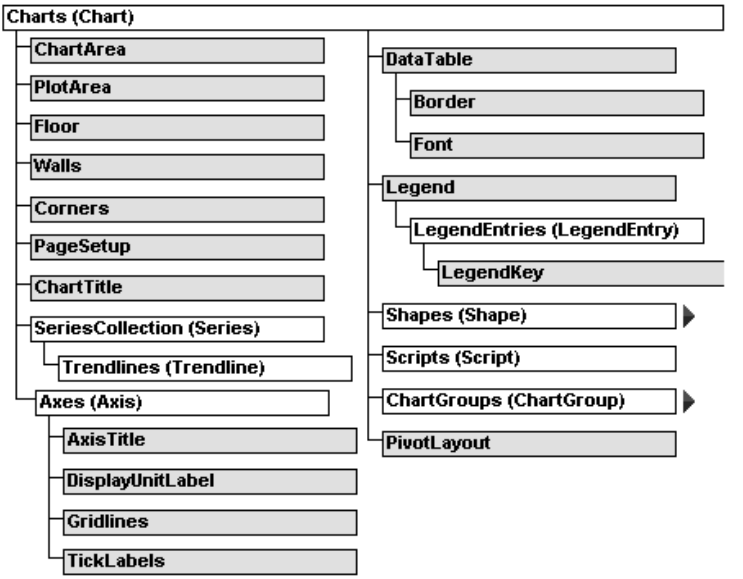


Рис. 3.3. Объект `Chart`
и непосредственно подчиняющиеся ему объекты

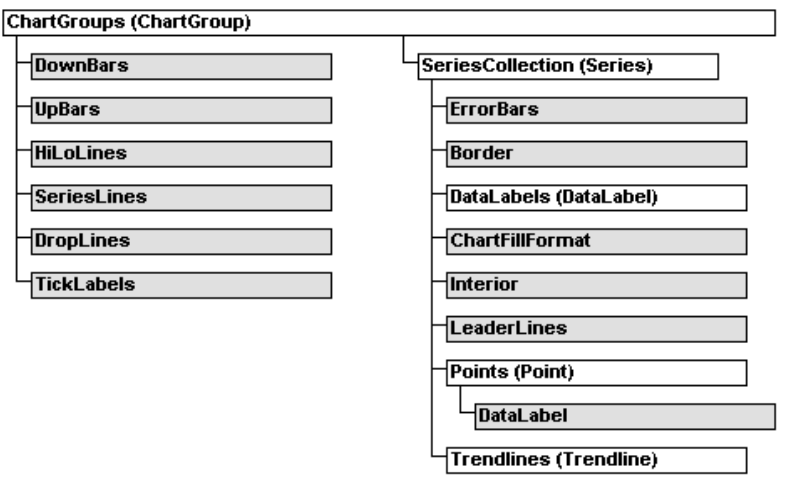


Рис. 3.4. Объект `ChartGroup`
и непосредственно подчиняющиеся ему объекты

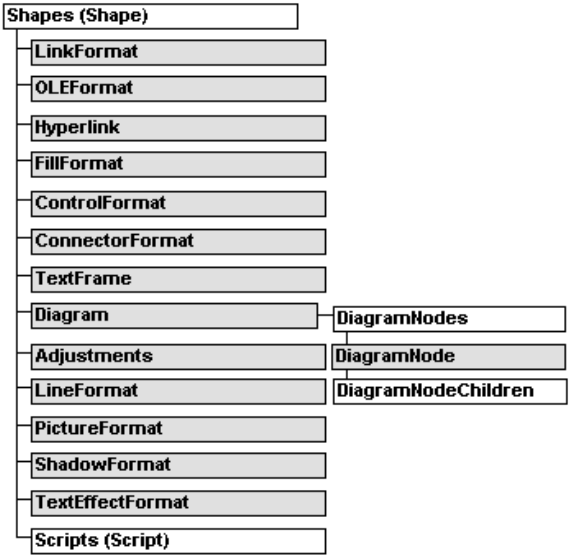


Рис. 3.5. Объект `Shape` и непосредственно подчиняющиеся ему объекты

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

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

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

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

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

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

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

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

Объект *Application*

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

Свойства объекта *Application*

Объект `Application`, благодаря обширной коллекции свойств, позволяет программно установить значения многих опций окна **Параметры**, отображаемого при выборе команды **Сервис | Параметры**. Кроме того, он обеспечивает доступ к объектам верхнего уровня типа `ActiveCell`, `ActiveSheet` и т. д. В последующих разделах на содержательных примерах подробно рассмотрены следующие свойства объекта `Application`:

```
ActiveCell
ActiveChart
ActivePrinter
ActiveSheet
ActiveWorkbook
AddIns
AlertBeforeOverwriting
Assistant
AutoCorrect
AutoFormatAsYouTypeReplaceHyperlinks
AutomationSecurity
AutoRecover
CalculateBeforeSave
Calculation
CalculationState
Caption
CellDragAndDrop
Cells
Charts
Columns
Cursor
DataEntryMode
DefaultFilePath
DefaultSaveFormat
DefaultWebOptions
Dialogs
DisplayAlerts
DisplayCommentIndicator
DisplayFormulaBar
DisplayFullScreen
DisplayScrollBars
DisplayStatusBar
EditDirectlyInCell
EnableCancelKey
FileSearch
FoundFiles
Height
Left
LibraryPath
MemoryFree
MemoryTotal
MemoryUsed
Names
OperatingSystem
OrganizationName
Path
ProductCode
RecentFiles
```

ReferenceStyle
ScreenUpdating
Selection
Sheets
StatusBar
ThisWorkbook
Top
UsableHeight
UsableWidth
UserName
Version
Width
Windows
WindowState
WorksheetFunction

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

Свойства ActiveWorkbook, ActiveSheet, ActiveCell, ActiveChart, ActivePrinter объекта Application возвращают активную рабочую книгу, лист, ячейку, диаграмму и принтер. Объект ActiveCell содержится в ActiveSheet, а объекты ActiveSheet и ActiveChart в ActiveWorkbook. В следующем примере (листинг 3.1) в активной ячейке устанавливается красный цвет фона с зеленым полужирным шрифтом и в нее вводится строка текста Май. Активный рабочий лист переименовывается в Отчет за май, а цвет его ярлыка назначается желтым.

Листинг 3.1. Ссылка на активный рабочий лист и ячейку

```
Sub DemoActives()
   With ActiveSheet
      .Tab.ColorIndex = 27 ' Желтый цвет
      .Name = "Отчет за май"
   End With
   With ActiveCell
      .Font.Bold = True
      .Font.Color = vbGreen
      ' vbGreen — встроенная константа, задающая зеленый цвет
      .Value = "Май"
      .Interior.Color = vbRed
      ' vbRed — встроенная константа, задающая красный цвет
   End With
End Sub
```

Инсталлированные надстройки

Свойство AddIns объекта Application возвращает семейство инсталлированных надстроек. Например, следующий код (листинг 3.2) проверяет, уста-

новлена ли надстройка Analysis ToolPak — VBA. Напомним, что надстройку можно инсталлировать, выбрав команду **Сервис | Надстройки**.

Листинг 3.2. Проверка, установлена ли указанная надстройка

```
If Application.AddIns( _
   "Analysis ToolPak — VBA").Installed = True Then
   MsgBox "Надстройка Analysis ToolPak — VBA " & _
      "инсталлирована"
Else
   MsgBox "Надстройка Analysis ToolPak — VBA " & _
      "не инсталлирована"
End If
```

Предупреждение перед копированием значения в заполненную ячейку

Свойство `AlertBeforeOverwriting` объекта `Application` позволяет управлять отображением диалогового окна с предупреждением при завершении операции drag-and-drop в непустой ячейке. Это свойство программирует установку или снятие флажка **Предупреждать перед перезаписью ячеек** на вкладке **Правка** окна **Параметры** отображаемого при выборе команды **Сервис | Параметры**. Например, следующая процедура, обрабатывающая событие `Open` рабочей книги, отменяет отображение диалогового окна с предупреждением при завершении операции drag-and-drop в непустой ячейке (листинг 3.3).

Листинг 3.3. Отмена отображения диалогового окна с предупреждением при завершении операции drag-and-drop в непустой ячейке. Модуль ЭтаКнига

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

Отображение помощника при выборе ячейки

Свойство `Assistant` объекта `Application` возвращает объект `Assistant`, который инкапсулирует в себе данные, свойства и методы, ассоциированные с помощником MS Office. Например, следующая процедура, обрабатывающая событие `SelectionChange` рабочего листа, показывает, как можно обеспечить отображение помощника при щелчке на ячейке **A1** рабочего листа и как его можно скрыть при щелчке на любой другой ячейке (листинг 3.4).

В этой процедуре используются следующие свойства и методы объекта Assistant:

- свойство On устанавливает, является ли помощник достижимым для пользователя. В данном примере помощник дееспособен лишь при выборе ячейки **A1** рабочего листа, т. е. только в этом случае запрос к справочной системе будет производиться через помощника, в то время как при выборе любой другой ячейки этот запрос реализуется через окно **Справка Microsoft Excel**;
- свойство Visible отображает или скрывает помощника;
- свойство Animation задает анимацию, ассоциированную с помощником. В данном случае она определяется встроенной константой msoAnimationThinking, которая обеспечивает прокручивание клипа с "думающим" помощником;
- метод Move задает координаты верхнего левого угла помощника.

Листинг 3.4. Отображение помощника при выборе ячейки. Модуль рабочего листа

```
Private Sub Worksheet_SelectionChange(ByVal Target As Range)
   If Target.Address = "$A$1" Then
      With Assistant
         .On = True
         .Visible = True
         .Move xLeft:=400, yTop:=300
         .Animation = msoAnimationThinking
      End With
   Else
      With Assistant
         .On = False
         .Visible = False
      End With
   End If
End Sub
```

Автозамена

Свойство AutoCorrect объекта Application возвращает объект AutoCorrect, который позволяет управлять автозаменой на рабочем листе. Свойства этого объекта программируют значение параметров, устанавливаемых в окне **Автозамена**, отображаемом при выборе команды **Сервис | Параметры автозамены**. В табл. 3.1 перечислены свойства объекта AutoCorrect.

Таблица 3.1. Свойства объекта `AutoCorrect`

Свойство	Описание
`CapitalizeNamesOfDays`	Автоматический перевод в верхний регистр названий месяцев
`CorrectCapsLock`	Автоматическая коррекция случайного нажатия клавиши <Caps Lock>
`CorrectSentenceCap`	Автоматический перевод в верхний регистр первой буквы предложения
`DisplayAutoCorrectOptions`	Управляет отображением кнопки **Параметры автозамены**
`ReplaceText`	Управляет автоматической заменой при вводе текста из списка автозамены
`ReplacementList`	Список автозамены. Добавление и удаление элементов из этого списка производится методами `AddReplacement` и `DeleteReplacement`.
`TwoInitialCapitals`	Исправление двух прописных букв в начале слова

Например, в следующем коде (листинг 3.5) первая процедура обрабатывает событие `Open` рабочей книги и обеспечивает добавление в список автозамены двух новых элементов. А именно `спб` будет при вводе автоматически заменяться на `Санкт-Петербург`, а `мск` — на `Москва`. Вторая процедура обрабатывает событие `BeforeClose`, генерируемое при закрытии рабочей книги. Она реализует исключение этих двух элементов из списка автозамены.

Листинг 3.5. Автозамена. Модуль `ЭтаКнига`

```
Private Sub Workbook_Open()
   Application.AutoCorrect.AddReplacement What:="спб", _
                                          Replacement:= "Санкт-Петербург"
   Application.AutoCorrect.AddReplacement What:="мск", _
                                          Replacement:="Москва"
   With Application.AutoCorrect
      .TwoInitialCapitals = False
      .CorrectSentenceCap = False
      .CapitalizeNamesOfDays = False
      .CorrectCapsLock = True
      .ReplaceText = True
      .DisplayAutoCorrectOptions = True
   End With
End Sub
```

```
Private Sub Workbook_BeforeClose(Cancel As Boolean)
   Application.AutoCorrect.DeleteReplacement What:="спб"
   Application.AutoCorrect.DeleteReplacement What:="мск"
End Sub
```

Замена текста гиперссылкой

Свойство `AutoFormatAsYouTypeReplaceHyperlinks` объекта `Application` управляет автоматической заменой текста гиперссылкой. Например, следующая инструкция запрещает такую замену, и вводимый текст не будет заменяться соответствующей гиперссылкой.

```
Application.AutoFormatAsYouTypeReplaceHyperlinks = False
```

Автосохранение

Свойство `AutoRecover` объекта `Application` возвращает объект `AutoRecover`, который инкапсулирует в себе свойства, позволяющие управлять автосохранением (табл. 3.2).

Таблица 3.2. *Свойства объекта* `AutoRecover`

Свойство	Описание
`Enabled`	Включает или отключает автосохранение
`Path`	Возвращает или устанавливает каталог, в который производится автосохранение
`Time`	Возвращает или устанавливает число минут, через которое производится автосохранение

Например, следующая процедура (листинг 3.6) в качестве параметров автосохранения устанавливает корневой каталог диска C: и время — 10 мин.

Листинг 3.6. Установка параметров автосохранения

```
Sub AutoRecover()
   Application.AutoRecover.Path = "C:\"
   Application.AutoRecover.Time = 10
End Sub
```

Управление уровнем безопасности

Свойство `AutomationSecurity` объекта `Application` позволяет управлять уровнем безопасности при открытии рабочей книги.

Это свойство возвращает одну из следующих констант MsoAutomationSecurity:

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

Следующая процедура (листинг 3.7) позволяет, используя диалоговое окно **Open**, открыть рабочую книгу с автоматическим отключением макросов. Для вызова диалогового окна **Open** применено свойство FileDialog объекта Application.

Листинг 3.7. Открытие рабочей книги с автоматическим отключением макросов

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

Задание режима вычисления

Свойство Calculation объекта Application устанавливает режим вычислений в ячейках рабочего листа. Допустимым значением является одна из следующих констант XlCalculation:

- xlCalculationAutomatic — автоматически;
- xlCalculationManual — вручную;
- xlCalculationSemiautomatic — автоматически, кроме таблиц.

Форсирование вычислений перед закрытием рабочей книги

Свойство CalculateBeforeSave объекта Application управляет форсированием вычислений перед закрытием рабочей книги, если значение свойства Calculation задано равным xlCalculationManual. Установку значения этого

свойства разумно расположить в процедуре обработки события BeforeClose рабочей книги, как показано в следующем коде (листинг 3.8).

Листинг 3.8. Форсирование вычислений перед закрытием рабочей книги

```
Private Sub Workbook_BeforeClose(Cancel As Boolean)
   Application.Calculation = xlCalculationManual
   Application.CalculateBeforeSave = True
End Sub
```

Проверка состояния вычисления

Проверку того, продолжает ли в данный момент времени MS Excel производить вычисления, можно осуществить при помощи свойства CalculationState объекта Application, допустимыми значениями которого являются следующие константы XlCalculationState: xlCalculating, xlDone и xlPending.

Управление перетаскиванием ячеек

Свойство CellDragAndDrop объекта Application управляет возможностью перетаскивания ячеек. Например, следующий код (листинг 3.9) при открытии рабочей книги отключает операции перетаскивания ячеек, а при ее закрытии — включает.

Листинг 3.9. Управление операцией drag-and-drop. Модуль ЭтаКнига

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

Private Sub Workbook_BeforeClose(Cancel As Boolean)
   Application.CellDragAndDrop = True
End Sub
```

Диапазон ячеек

Свойство Cells объекта Application возвращает объект типа Range, т. е. диапазон ячеек. У этого свойства допустимо два варианта синтаксиса — с параметрами и без них. Без параметров свойство возвращает все ячейки диапазона, а с параметрами — ячейку, стоящую в указанных строке и столбце. В следующем демонстрационном примере (листинг 3.10) у всех ячеек активного рабочего листа устанавливаются атрибуты шрифта — название (свойство Name), цвет (свойство ColorIndex) и размер (свойство Size), кроме того, в ячейку **A1** вводится значение Алиса, а в ячейку **A2** — в стране чудес.

Листинг 3.10. Работа с диапазоном ячеек

```
Sub DemoCells()
   With Application.Cells.Font
      .Name = "Arial"
      .ColorIndex = 3
      .Size = 10
   End With
   Application.Cells(1, 1).Value = "Алиса"
   Application.Cells(1, 2).Value = "в стране чудес"
End Sub
```

Столбцы и строки рабочего листа

Свойство `Columns` объекта `Application` возвращает объект типа `Range`, который в данном контексте представляет собой набор столбцов. У этого свойства допустимо два варианта синтаксиса — с параметром и без него. Без параметра свойство возвращает все столбцы диапазона, а с параметром — столбец со специфицированным номером.

Свойство `Rows` аналогично свойству `Columns`, но возвращает не столбцы, а строки.

Следующий код показывает, как, используя свойство `Columns`, можно во всех ячейках указанного столбца задать стиль шрифта (в данном случае — полужирный), а также ввести одно и то же значение (1) за одну операцию.

```
Sub DemoColumns1()
   Application.Columns(1).Font.Bold = True
   Application.Columns(1).Value = 1
End Sub
```

Листы с диаграммами

Свойство `Charts` объекта `Application` возвращает семейство `Charts` всех листов с диаграммами, расположенных в данном приложении. В качестве примера приведем процедуру, которая удаляет из приложения все листы с диаграммами (листинг 3.11). В этом коде используется свойство `Count` семейства `Charts`, возвращающее число элементов этого семейства, и метод `Delete`, удаляющий элементы из семейства.

Листинг 3.11. Удаление всех листов с диаграммами

```
Sub DeleteCharts()
   If Application.Charts.Count > 0 Then
```

```
      Application.Charts.Delete
   End If
End Sub
```

Рабочая книга, в которой выполняется данный макрос

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

Установка заголовка окна MS Excel

Свойство `Caption` объекта `Application` возвращает или устанавливает текст из заголовка главного окна MS Excel. Установка значения свойства равным `Empty` (пустая строка) возвращает заголовок, используемый по умолчанию. В следующем примере (листинг 3.12) первая процедура обрабатывает событие `Open` рабочей книги и устанавливает в качестве заголовка окна приложения текст `Отчет за 2002 год`, а вторая — обрабатывает событие `BeforeClose` рабочей книги и возвращает окну имя, используемое по умолчанию, т. е. `Microsoft Excel` (рис. 3.6).

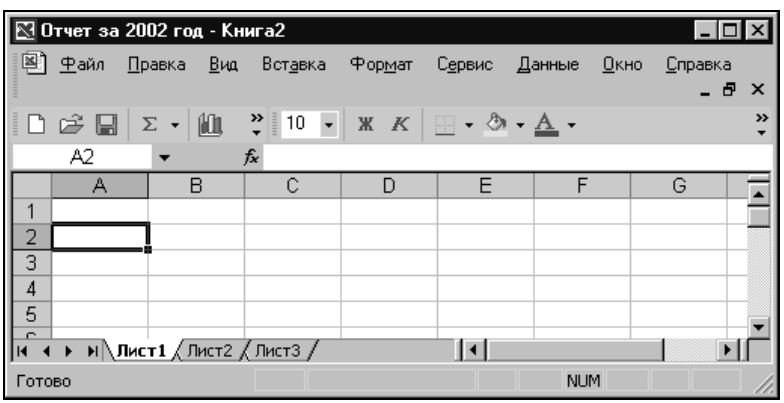


Рис. 3.6. Окно MS Excel с пользовательским заголовком

Листинг 3.12. Установка заголовка окна MS Excel. Модуль `ЭтаКнига`

```
Private Sub Workbook_Open()
   Application.Caption = "Отчет за 2002 год"
End Sub
```

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

Установка курсора

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

- `xlDefault` (курсор, используемый по умолчанию);
- `xlWait` (песочные часы);
- `xlNorthwestArrow` (стрелка в северо-западном направлении);
- `xlIBeam` (вертикальная полоска).

Если вы используете нестандартный курсор, например, для уведомления пользователя, что приложение не зависло, а идут вычисления, по завершении вычислений не забудьте установить курсор, используемый по умолчанию в системе. Например, следующий код (листинг 3.13) находит сумму обратных величин квадратов целых чисел. Так как суммируется достаточно много чисел, на время суммирования курсор принимает вид песочных часов. По завершении суммирования курсору назначается вид, используемый по умолчанию.

Листинг 3.13. Изменение курсора на время вычислений

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

Видоизменять курсор можно не только при вычислениях, но и при любых других действиях, например при выборе ячейки. В следующем примере при выборе ячейки **A1** курсор превращается в указатель, направленный на северо-запад, а при выборе любой другой — он принимает тот вид, который используется в приложении по умолчанию (листинг 3.14).

Листинг 3.14. Изменение курсора после выбора специфицированной ячейки. Модуль рабочего листа

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

Блокировка листа на время ввода данных

Свойство `DataEntryMode` объекта `Application` обеспечивает блокировку листа на время ввода данных. Допустимые значения:

- `xlOn` — режим блокировки включен, нажатие на клавишу <Esc> его выключает;
- `xlOff` — режим блокировки выключен;
- `xlStrict` — режим блокировки включен, и нажатие на клавишу <Esc> не может его выключить.

Каталог, используемый по умолчанию MS Excel при открытии файлов

Свойство `DefaultFilePath` объекта `Application` возвращает каталог, используемый по умолчанию MS Excel при открытии файлов.

Формат, используемый по умолчанию при сохранении рабочей книги

Свойство `DefaultSaveFormat` объекта `Application` возвращает и устанавливает формат, используемый по умолчанию MS Excel при сохранении книги.

Установка опций книги, когда она используется как Web-страница

Свойство `DefaultWebOptions` объекта `Application` возвращает объект `DefaultWebOptions`, который инкапсулирует в себе данные, используемые при работе с книгой, когда она открыта как Web-страница.

Семейство встроенных диалоговых окон

Свойство `Dialogs` возвращает семейство `Dialogs` всех встроенных диалоговых окон. Параметр этого семейства идентифицирует окно. Метод `Show` ото-

бражает его на экране, а параметры этого метода задают параметры, специфицируемые в отображаемом окне. Метод `Show` возвращает значение `True`, если задача, поставленная в отображаемом окне, была выполнена успешно. Например, следующий код (листинг 3.15) отображает окно **Открытие документа** для открытия книги C:\Test.xls.

Листинг 3.15. Открытие книги при помощи встроенного окна

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

Отображение встроенных предупреждений о работе программы

Свойство `DisplayAlerts` объекта `Application` устанавливает, надо ли отображать встроенные предупреждения о работе программы. Например, следующая процедура (листинг 3.16), обрабатывающая событие `Open` объекта `Workbook`, отменяет отображение предупреждений при работе с книгой.

Листинг 3.16. Отмена отображения предупреждения о необходимости сохранить документ

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

Отображение примечаний

Свойство `DisplayCommentIndicator` объекта `Application` позволяет управлять стилем отображения примечаний. Допустимыми значениями этого свойства являются следующие константы `XlCommentDisplayMode`:

- `xlNoIndicator` — нет индикатора;
- `xlCommentIndicatorOnly` — только индикатор;
- `xlCommentAndIndicator` — примечание и индикатор.

В демонстрационном примере (листинг 3.17) при открытии рабочей книги к ячейкам **A1** и **A4** добавляется по примечанию. При выборе ячейки **A1** отображаются как примечания, так и их индикаторы, а при выборе любой другой ячейки — они скрываются (рис. 3.7).

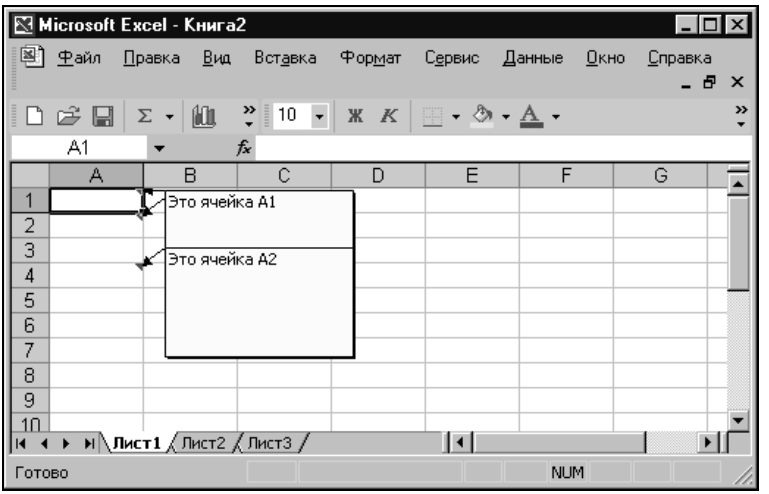


Рис. 3.7. Управление отображением примечаний и их индикаторов

Листинг 3.17. Управление отображением примечаний и их индикаторов. Модуль ЭтаКнига

```
Private Sub Workbook_Open()
   Worksheets(1).Range("A1").ClearComments
   Worksheets(1).Range("A1").AddComment
   Worksheets(1).Range("A1").Comment.Visible = True
   Worksheets(1).Range("A1").Comment.Text Text:="Это ячейка A1"
   Worksheets(1).Range("A4").ClearComments
   Worksheets(1).Range("A4").AddComment
   Worksheets(1).Range("A4").Comment.Visible = True
   Worksheets(1).Range("A4").Comment.Text Text:="Это ячейка A2"
End Sub

Private Sub Workbook_SheetSelectionChange(ByVal Sh As Object, _
                                          ByVal Target As Range)
   If Sh.Name = Worksheets(1).Name Then
      If Target.Address = "$A$1" Then
         Application.DisplayCommentIndicator = xlCommentAndIndicator
```

```
      Else
         Application.DisplayCommentIndicator = xlNoIndicator
      End If
   End If
End Sub
```

Отображение строки формул, полосы прокрутки и строки состояния

Свойства `DisplayFormulaBar`, `DisplayScrollBars` и `DisplayStatusBar` объекта `Application` устанавливают, надо ли отображать строку формул, полосу прокрутки и строку состояния.

Задание текста, выводимого в строке состояния

Свойство `StatusBar` объекта `Application` возвращает или устанавливает текст, выводимый в строке состояния. Следующая процедура (листинг 3.18), обрабатывающая событие `SheetSelectionChange` рабочей книги, обеспечивает вывод в строку состояния адрес выбранного диапазона (рис. 3.8).

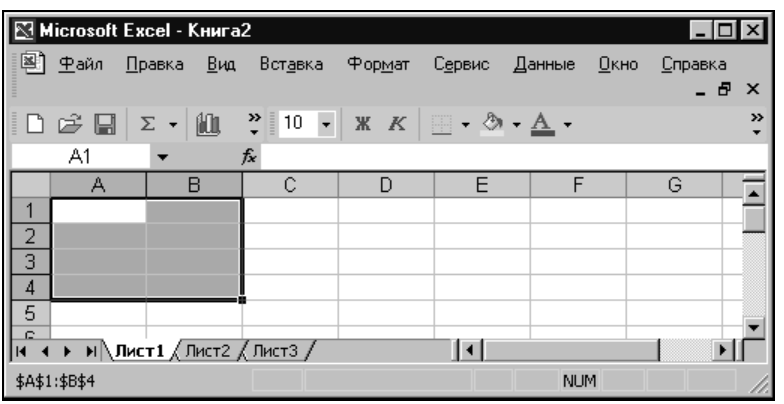


Рис. 3.8. Задание текста, выводимого в строке состояния

Листинг 3.18. Задание текста, выводимого в строке состояния. Модуль ЭтаКнига

```
Private Sub Workbook_SheetSelectionChange(ByVal Sh As Object, _
                                          ByVal Target As Range)
   Application.StatusBar = Target.Address
End Sub
```

Полноэкранное отображение рабочего листа

Свойство `DisplayFullScreen` объекта `Application` управляет отображением рабочего листа в полноэкранном виде. Например, следующий код (листинг 3.19) обеспечивает полноэкранное отображение окна при щелчке правой кнопкой мыши на ячейке рабочего листа.

Листинг 3.19. Полноэкранное отображение рабочего листа. Модуль ЭтаКнига

```
Private Sub Workbook_SheetBeforeRightClick(ByVal Sh As Object, _
                                           ByVal Target As Range, _
                                           Cancel As Boolean)
   Cancel = True
   Application.DisplayFullScreen = True
End Sub
```

Управление действием операционной системы по нажатии комбинации клавиш <Ctrl>+<Break>

Свойство `EnableCancelKey` объекта `Application` задает действие операционной системы по нажатии комбинации клавиш <Ctrl>+<Break>, используемой для прерывания выполнения программы. Допустимые значения:

- `xlDisabled` — запрет прерывания программы;
- `xlInterrupt` — разрешено прерывание процедуры;
- `xlErrorHandler` — прерывание воспринимается как ошибка.

Установка высоты и ширины окна приложения

Свойства `Height` и `Width` объекта `Application` устанавливают высоту и ширину окна приложения. Эти свойства не могут быть заданы, если значение свойства `WindowState` равно `xlMaximized`.

Установка координаты левого верхнего угла окна приложения

Свойства `Left` и `Top` объекта `Application` устанавливают расстояние от левой границы окна приложения до левого и верхнего края экрана. Кроме того, свойство `Right` позволяет задать расстояние от правой границы окна приложения до правого края экрана. Например, следующий код при открытии рабочей книги устанавливает ее размеры и местоположение (листинг 3.20).

Листинг 3.20. Установка размеров и местоположения окна приложения. Модуль ЭтаКнига

```
Private Sub Workbook_Open()
   Application.WindowState = xlNormal
   Application.Width = 300
   Application.Height = 200
   Application.Top = 50
   Application.Left = 10
End Sub
```

Максимальные размеры рабочей книги

Свойства `UsableHeight` и `UsableWidth` объекта `Application` возвращают максимально допустимые высоту и ширину рабочей книги. Следующий код (листинг 3.21) при открытии книги устанавливает размеры активного окна в половину максимально допустимых (рис. 3.9).

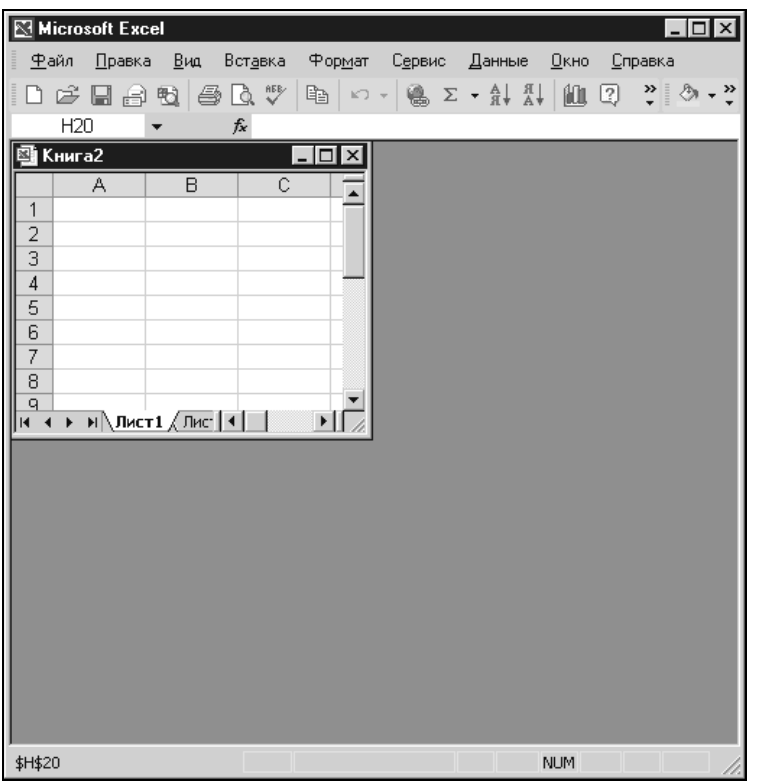


Рис. 3.9. Установка размеров рабочей книги

Листинг 3.21. Рабочая книга вполовину допустимых максимальных размеров. Модуль ЭтаКнига

```
Private Sub Workbook_Open()
   With ActiveWindow
      .WindowState = xlNormal
      .DisplayVerticalScrollBar = True
      .DisplayHeadings = True
      .Top = 0
      .Left = 0
      .Height = Application.UsableHeight / 2
      .Width = Application.UsableWidth / 2
   End With
End Sub
```

Нахождение пути к файлу Excel.exe и к библиотечному файлу

Свойства `Path` и `LibraryPath` объекта `Application` возвращают путь к файлу Excel.exe и к библиотечному файлу.

```
Msgbox Application.Path & vbCr & Application.LibraryPath
```

Управление возможностью редактирования данных в ячейке

Свойство `EditDirectlyInCell` объекта `Application` управляет возможностью редактирования данных в ячейке. Например, следующая инструкция запрещает пользователю редактировать данные в ячейке, но не вводить их в нее. Он может это делать только с помощью строки формул.

```
Application.EditDirectlyInCell = False
```

Поиск файлов

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

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

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

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

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

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

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

msoSortByFileName	msoSortByNone
msoSortByFileType	msoSortBySize
msoSortByLastModified	

- *`SortOrder`* — необязательный параметр, определяющий порядок сортировки. Допустимые значения определяются следующими `MsoSortOrder`-константами: `msoSortOrderAscending`, `msoSortOrderDescending`;
- *`AlwaysAccurate`* — необязательный параметр, принимающий логические значения и определяющий, должны ли при поиске просматриваться те файлы, которые были добавлены, модифицированы или удалены после последней модификации файлового индекса.

В приводимом ниже примере (листинг 3.22) производится поиск всех рабочих книг в корневом каталоге диска C:. Если таковые имеются, то отображается сообщение с указанием их количества и полным списком имен. Если же их нет, то пользователь также об этом информируется.

Листинг 3.22. Поиск рабочих книг в корневом каталоге диска C:

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

Свободная, общая и используемая память

Свойства `MemoryFree`, `MemoryTotal` и `MemoryUsed` объекта `Application` возвращают свободную, общую и используемую системой память. Следующий код (листинг 3.23) демонстрирует использование этих свойств.

Листинг 3.23. Свободная, общая и используемая память

```
Sub MemoryInfo()
   Dim str As String
   str = "MemoryFree: " & Application.MemoryFree & vbCr
   str = str + "MemoryTotal: " & Application.MemoryTotal & vbCr
   str = str + "MemoryUsed: " & Application.MemoryUsed
   MsgBox str
End Sub
```

Имя операционной системы

Свойство `OperatingSystem` объекта `Application` возвращает имя установленной операционной системы. Например, следующий код отображает диалоговое окно с сообщением `Windows (32-bit) NT 5.01`.

```
MsgBox Application.OperatingSystem
```

Название организации, для которой зарегистрирована операционная система, имя пользователя и код продукта

Свойства `OrganizationName`, `UserName` и `ProductCode` объекта `Application` возвращают название организации, для которой зарегистрирована операционная система, имя пользователя и код продукта.

Семейство всех имен активной рабочей книги

Свойство `Names` объекта `Application` возвращает семейство `Names` всех имен активной рабочей книги. Как и у любого семейства, свойство `Count` семейства `Names` возвращает число его элементов. Свойство `Name` возвращает имя конкретного элемента семейства, а свойство `RefersToRange` — объект `Range`, для которого установлено имя. В следующем примере у первого листа рабочей книги определяется семейство всех его имен, а затем они, так же как и ссылки на те диапазоны, которым они были присвоены, выводятся последовательно в ячейки первого и второго столбцов рабочей книги (листинг 3.24).

Листинг 3.24. Семейство всех имен активной рабочей книги

```
Sub DemoNames()
   Dim nms As Names
   Dim wks As Worksheet
   Dim i As Integer
   Set nms = ActiveWorkbook.Names
   Set wks = Worksheets(1)
   If nms.Count = 0 Then
      MsgBox "Нет имен на рабочем листе" & wks.Name
   Else
      For i = 1 To nms.Count
         wks.Cells(i, 1).Value = nms(i).Name
         wks.Cells(i, 2).Value = nms(i).RefersToRange.Address
      Next
   End If
End Sub
```

Семейство недавно использованных файлов

Свойство `RecentFiles` объекта `Application` возвращает семейство `RecentFiles` объектов `RecentFile`, каждый из которых инкапсулирует в себе информацию о недавно использованных файлах, имена которых отображаются в нижней части выпадающего меню **Файл**. Метод `Add` позволяет добавить новый эле-

мент в семейство. Кроме того, в семействе имеются свойства, перечисленные в табл. 3.3.

Таблица 3.3. *Свойства семейства* `RecentFiles`

Свойство	Описание
`Count`	Возвращает число элементов семейства
`Maximum`	Возвращает и устанавливает максимально допустимое число элементов семейства. Может быть любым числом из диапазона от 0 до 9
`Path`	Возвращает элемент семейства, специфицированный по его индексу

У объекта `RecentFile` имеются следующие свойства и методы, перечисленные в табл. 3.4 и 3.5.

Таблица 3.4. *Свойства объекта* `RecentFile`

Свойство	Описание
`Index`	Возвращает индекс элемента в семействе
`Name`	Возвращает имя файла
`Path`	Возвращает путь к файлу

Таблица 3.5. *Методы объекта* `RecentFile`

Метод	Описание
`Delete`	Удаляет элемент из семейства
`Open`	Открывает книгу

В следующем примере первая инструкция устанавливает максимальное число в списке недавно использованных файлов, а вторая — открывает второй из этих файлов:

```
Application.RecentFiles.Maximum = 3
Application.RecentFiles.Item(2).Open
```

Установка стиля ссылок на диапазоны

Свойство `ReferenceStyle` объекта `Application` устанавливает стиль ссылок (`A1` или `R1C1`) на диапазон. Допустимые значения определяются следующими `XlReferenceStyle`-константами: `xlA1` и `xlR1C1`.

Обновление изображения во время выполнения программы

Свойство `ScreenUpdating` объекта `Application` устанавливает, надо ли обновлять изображение во время выполнения программы. Задание этому свойству значения `False` ускоряет выполнение программы. В конце программы необходимо свойству `ScreenUpdating` присвоить значение `True`.

Ссылка на выбранный объект

Свойство `Selection` возвращает ссылку на выбранный объект, например, рабочий лист, диапазон, диаграмму. Таким образом, тип элемента, возвращаемого свойством `Selection`, зависит от того, что было выбрано. Например, в следующем коде при помощи метода `Select` сначала выбирается диапазон **A1:A10**, затем свойство `Selection` возвращает ссылку на этот диапазон и, в заключение, методом `Clear` этот диапазон очищается.

```
Worksheets(1).Range("A1:A10").Select
Selection.Clear
```

Семейство листов активной рабочей книги

Свойство `Sheets` возвращает семейство листов активной рабочей книги. В семействе `Sheets` имеются свойства и методы, перечисленные в табл. 3.6 и 3.7.

Таблица 3.6. *Свойства семейства* `Sheets`

Свойство	Описание
`Count`	Возвращает число элементов семейства
`Item`	Возвращает элемент семейства

Таблица 3.7. *Методы семейства* `Sheets`

Метод	Описание
`Add`	Создает новый лист: `Add(`*`Before, After, Count, Type`*`)` ♦ *`Before`* — необязательный параметр, специфицирующий лист, перед которым будет добавлен новый; ♦ *`After`* — необязательный параметр, специфицирующий лист, после которого будет добавлен новый;

Таблица 3.7 *(продолжение)*

Метод	Описание
(прод.)	♦ *Count* — необязательный параметр, указывающий число добавляемых листов. По умолчанию его значение равно единице; ♦ *Type* — необязательный параметр, задающий тип листа. Его допустимым значением может быть одна из следующих XlSheetType-констант: xlWorksheet, xlChart, xlExcel4MacroSheet и xlExcel4IntlMacroSheet. По умолчанию подразумевается значение xlWorksheet
Copy	Копирует лист: Copy(*Before*, *After*) ♦ *Before* — необязательный параметр, специфицирующий лист, перед которым будет вставлен данный; ♦ *After* — необязательный параметр, специфицирующий лист, после которого будет вставлен данный
Delete	Удаляет лист
Move	Перемещает лист: Move(*Before*, *After*) ♦ *Before* — необязательный параметр, специфицирующий лист, перед которым будет вставлен данный; ♦ *After* — необязательный параметр, специфицирующий лист, после которого будет вставлен данный
PrintOut	Печать листа или группы листов: PrintOut(*From*, *To*, *Copies*, *Preview*, ↳*ActivePrinter*, *PrintToFile*, *Collate*, ↳*PrToFileName*) ♦ *From* — необязательный параметр, специфицирующий номер листа, с которого начинается печать группы специфицированных рабочих листов; ♦ *To* — необязательный параметр, специфицирующий номер последнего листа из группы печатаемых листов; ♦ *Copies* — необязательный параметр, задающий число печатаемых копий; ♦ *Preview* — необязательный параметр, причем, если его значение равно True, то перед осуществлением печати MS Excel отображает окно предварительного просмотра, а если False — то печать происходит непосредственно без отображения этого окна;

Таблица 3.7 (окончание)

Метод	Описание
(прод.)	♦ *ActivePrinter* — необязательный параметр, задающий имя активного принтера; ♦ *PrintToFile* — необязательный параметр, причем, если его значение равно True, то печать производится в файл; ♦ *Collate* — необязательный параметр, причем, если его значение равно True, MS Excel разбирает документ по печатаемым копиям; ♦ *PrToFileName* — необязательный параметр, указывающий имя выходного файла, если значение параметра *PrintToFile* установлено равным True
PrintPreview	Отображение окна предварительного просмотра
Select	Выбирает специфицированный лист
FillAcrossSheets	Копирует указанный диапазон в другие рабочие листы данной коллекции: FillAcrossSheets(*Range*, *Type*) ♦ *Range* — обязательный параметр, дающий ссылку на копируемый диапазон; ♦ *Type* — необязательный параметр, специфицирующий то, как диапазон будет копироваться. Допустимые значения определяются следующими XlFillWith-константами: xlFillWithAll, xlFillWithContents и xlFillWithFormats

Печать активного листа

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

```
ActiveSheet.PrintOut
```

Предварительный просмотр книги

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

```
Application.Sheets.PrintPreview
```

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

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

Доступ из кода к функциям рабочего листа

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

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

Примечание

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

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

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

Номер версии MS Excel

Свойство Version объекта Application возвращает номер текущей версии MS Excel. Используется для проверки, применяется ли корректная версия. Например, следующая инструкция проверяет версию MS Excel, и если она отлична от 10.0, то происходит выход из процедуры.

```
If Application.Version <> "10.0" Then Exit Sub
```

Семейство всех окон приложения

Свойство Windows возвращает семейство всех окон приложения. В следующем примере активизируется окно книги Отчет:

```
Application.Windows("Отчет.xls").Activate
```

Приведенная в листинге 3.25 процедура отображает список имен всех окон данного приложения.

Листинг 3.25. Список имен всех окон данного приложения

```
Sub ShowWindows()
   Dim w As Window
   Dim s As String
   For Each w In Application.Windows
      s = s & w.Caption & vbCr
   Next
   MsgBox s
End Sub
```

Управление состоянием окна

Свойство `WindowState` объекта `Application` устанавливает состояние окна. Допустимыми значениями являются следующие константы `XlWindowState`: `xlMaximized` — полноэкранное, `xlMinimized` — свернутое, `xlNormal` — стандартное.

Методы объекта *Application*

У объекта `Application` имеется большая коллекция методов, позволяющих производить различные действия — от конвертации метрических единиц измерения до создания таймера. В последующих разделах обсуждаются на содержательных примерах следующие основные методы объекта `Application`:

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

Запуск или активизация MS-приложения

Метод `ActivateMicrosoftApp` объекта `Application` запускает указанное MS-приложение, если оно еще не было запущено, или активизирует его в противном случае.

```
ActivateMicrosoftApp(index)
```

Здесь параметр *index* может быть одной из следующих констант XlMSApplication: xlMicrosoftWord, xlMicrosoftPowerPoint, xlMicrosoftMail, xlMicrosoftAccess, xlMicrosoftFoxPro, xlMicrosoftProject, xlMicrosoftSchedulePlus. Например, следующая инструкция запускает или активизирует MS Word.

```
Application.ActivateMicrosoftApp xlMicrosoftWord
```

Принудительное вычисление во всех открытых рабочих книгах

Метод Calculate объекта Application вызывает принудительное вычисление во всех открытых рабочих книгах, или специфицированном рабочем листе, или диапазоне. В следующем примере первая инструкция приводит к перерасчету во всех открытых книгах, вторая инструкция — к перерасчету на указанном рабочем листе активной рабочей книги, а третья — в указанном диапазоне:

```
Application.Calculate
Worksheets("Отчет").Calculate
Worksheets("Отчет ").Range("A1:C10").Calculate
```

Перевод сантиметров и дюймов в пункты

Методы CentimetersToPoints и InchesToPoints объекта Application переводят сантиметры и дюймы в пункты. Один пункт равен 1/72 дюйма или приблизительно 0,353 мм. В следующем примере диаграмма располагается на активном рабочем листе на расстоянии 3 см по горизонтали и 2 см по вертикали от левого верхнего угла ячейки **A1**.

```
With ActiveSheet.ChartObjects(1)
   .Left = Application.CentimetersToPoints(2)
   .Top = Application.CentimetersToPoints(3)
End With
```

Проверка правописания отдельного слова

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

```
CheckSpelling(Word, CustomDictionary, IgnoreUppercase)
```

- *Word* — обязательный параметр, используемый только с объектом Application, специфицирующий проверяемое слово;

- *CustomDictionary* — необязательный параметр, указывающий имя файла с пользовательским словарем;
- *IgnoreUppercase* — необязательный параметр, принимающий логические значения и определяющий, надо ли проверять правописание слов, набранных строчными буквами.

Например, следующий код (листинг 3.26) проверяет принадлежность указанного слова словарю.

Листинг 3.26. Проверка принадлежности указанного слова словарю

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

Перевод ссылки на диапазон из формата *A1* в формат *R1C1*

Метод ConvertFormula объекта Application переводит ссылки на ячейки в формуле из формата **A1** в формат **R1C1**, т. е. из относительной в абсолютную.

```
ConvertFormula(Formula, FromReferenceStyle, ToReferenceStyle, ToAbsolute,
↳RelativeTo)
```

- *Formula* — обязательный параметр, задающий строковое выражение с преобразуемой формулой;
- *FromReferenceStyle* — обязательный параметр, определяющий формат преобразуемой формулы. Допустимые значения: xlA1 и xlR1C1;
- *ToReferenceStyle* — необязательный параметр, устанавливающий формат, в который формула преобразуется. Допустимые значения: xlA1 и xlR1C1;
- *ToAbsolute* — необязательный параметр, задающий разновидность абсолютного формата (т. е. является абсолютной ссылка на строку, столбец или и то и другое), в который формула преобразуется. Допустимые значения: xlAbsolute, xlAbsRowRelColumn, xlRelRowAbsColumn и xlRelative;
- *RelativeTo* — необязательный параметр, задающий объект Range, состоящий из одной ячейки, относительно которой в формуле даются относительные ссылки.

В следующем коде (листинг 3.27) в ячейку **B1** вводится формула `=СУММ(A1:A10)`, а ячейку **B2** та же самая формула, но в абсолютном формате.

Листинг 3.27. Ввод формул в ячейки в разных форматах

```
Sub DemoConvertFormula()
   Dim f As String
   f = "=СУММ(A1:A10)"
   Range("B1").FormulaLocal = f
   Range("B2").FormulaLocal = Application.ConvertFormula(Formula:=f, _
                                          FromReferenceStyle:=xlA1, _
                                          toAbsolute:=xlAbsolute)
End Sub
```

Диалоговое окно ввода данных

Метод `InputBox` объекта `Application` выводит на экран диалоговое окно, содержащее сообщение, поле ввода, две кнопки — **OK** и **Отмена** (Cancel). Устанавливает режим ожидания ввода данных пользователем и нажатия кнопки, а затем возвращает введенное в поле ввода значение по нажатии на кнопку **OK**. При нажатии кнопки **Отмена** возвращается значение `False`. Метод `InputBox` отличен по своим функциональным возможностям от функции `InputBox`.

```
InputBox(Prompt, Title, Default, Left, Top, HelpFile,
↳HelpContextId, Type)
```

- *Prompt* — обязательный параметр, задающий строковое выражение, отображаемое как сообщение в диалоговом окне. Этот параметр позволяет выводить многострочное сообщение. Для разделения строки на несколько надо воспользоваться символами возврата каретки (`Chr(13)`), перевода строки (`Chr(10)`) или их комбинацией (`Chr(13) & Chr(10)`);
- *Title* — необязательный параметр, задающий строку заголовка диалогового окна. Если этот параметр опущен, то в строке заголовка отображается **Ввод** (Input);
- *Default* — необязательный параметр, задающий значение, отображаемое в поле ввода как используемое по умолчанию, если пользователь не введет другие данные. Если этот параметр опущен, то поле ввода изображается пустым;
- *Left*, *Top* — необязательные параметры, задающие расстояние по горизонтали и вертикали между левыми верхними углами диалогового окна и экрана;

- *HelpFile* — необязательный параметр, указывающий имя справочного файла, содержащего справочные сведения о данном диалоговом окне. Если этот параметр указан, то необходимо задать также значение параметра *HelpContextId*;
- *HelpContextId* — необязательный параметр, определяющий идентификатор соответствующего раздела справочной системы. Если этот параметр используется, то необходимо указать также значение параметра *HelpFile*;
- *Type* — необязательный параметр, специфицирующий тип возвращаемого значения. Если этот параметр опущен, то возвращается строковое выражение. Допустимые значения параметра перечислены в табл. 3.8. Разрешено использование суммы этих значений, например, значение параметра, равное 1 + 2, разрешает возврат как чисел, так и строк.

Таблица 3.8. *Значения параметра* *Type* *метода* *InputBox*

Значение	Описание
0	Формула
1	Число
2	Строка
4	Логическое значение (`True` или `False`)
8	Ссылка на ячейки, как на объект типа `Range`
16	Сообщение об ошибке типа `#N/A`
64	Массив значений

Ввод ссылки на ячейку

Метод `InputBox` позволяет производить ввод из диалогового окна ссылки на ячейку или диапазон ячеек. Для этого в качестве значения его параметра *Type* надо указать 8. В следующем коде (листинг 3.28) после нажатия кнопки **OK** в выбранную ячейку при помощи диалогового окна вводится строка `"Ося и Киса были здесь"` (рис. 3.10).

Листинг 3.28. Ввод ссылки на ячейку

```
Dim myCell As Range
Set myCell = Application.InputBox(Prompt:="Выберите ячейку", Type:=8)
myCell.Value = "Ося и Киса были здесь"
```

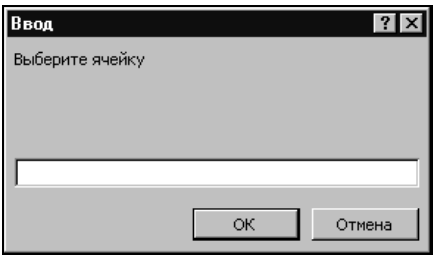


Рис. 3.10. Окно для ввода ссылки на ячейку

Ввод числа

Метод `InputBox` обеспечивает ввод из диалогового окна числовых данных. Для этого в качестве значения его параметра *`Type`* надо указать 1. В следующем коде (листинг 3.29) после нажатия кнопки **OK** введенное значение считывается, возводится в квадрат, и найденное значение отображается в диалоговом окне.

Листинг 3.29. Ввод числа

```
Dim r As Double
r = Application.InputBox(Prompt:="Введите число", _
                         Title:="Квадрат числа", Type:=1)
MsgBox "Квадрат числа: " & r ^ 2
```

Преобразование имени MS Excel в объект или значение

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

```
Evaluate(Name)
```

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

- ссылка формата **A1** на ячейку, которая рассматривается как абсолютная ссылка;
- диапазон;
- пользовательское имя;
- внешние ссылки. При этом надо использовать оператор `!` для указания ссылки на объект из другой книги. Например:

  `Evaluate("[Отчет.XLS]Май!A1").`

Симулирование вычисления арифметических выражений

Метод `Evaluate` позволяет по строковому представлению арифметического выражения находить его значение. Например, следующий оператор (листинг 3.30) отображает значение выражения `(1+3)/2+sin(2)`.

Листинг 3.30. Калькулятор

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

Простейший функциональный калькулятор

Обобщая идею, обсужденную в предыдущем разделе, используя метод `Evaluate`, легко создать простейший функциональный калькулятор, который возвращает значение введенного арифметического выражения (листинг 3.31). Для этого пользователю надо иметь средство ввода строкового представления арифметического выражения, что можно обеспечить методом `InputBox`. Метод `Evaluate` произведет требуемые вычисления, а диалоговое окно `MsgBox` обеспечит отображения результата.

Листинг 3.31. Простейший функциональный калькулятор

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

Симулирование ячеек рабочего листа

Метод `Evaluate` позволяет симулировать работу с ячейками или диапазонами рабочего листа без реального воплощения этих действий на рабочем листе. Например, в следующем коде (листинг 3.32) симулируется ввод в ячейку **A1** значения `25`, ввод в ячейку **A2** формулы `=A1^2` и считывание возвращаемого этой формулой значения.

Листинг 3.32. Симулирование ввода данных в ячейки и считывания из них значений

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

Ту же задачу можно решить в следующем коде (листинг 3.33), но здесь уже производится симулирование не операции ввода и вывода, а самих ячеек.

Листинг 3.33. Симулирование ячеек

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

Применение квадратных скобок при симулировании операций на рабочем листе

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

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

или

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

Отображение справки

Метод `Help` объекта `Application` отображает справку.

```
Help(HelpFile, HelpContextID)
```

- *HelpFile* — необязательный параметр, задающий имя HLP-файла. Если значение этого параметра не указано, то отображается файл справки Microsoft Excel;
- *HelpContextID* — необязательный параметр, указывающий номер раздела справки. Если значение этого параметра не указано, то отображается оглавление справки.

Например, следующая инструкция (листинг 3.34) вызывает справку к программе MS Notepad при щелчке правой кнопкой мыши на ячейке **A1**.

Листинг 3.34. Вызов справки. Модуль рабочего листа

```
Private Sub Worksheet_BeforeRightClick(ByVal Target As Range, _
      Cancel As Boolean)
   If Target.Address = "$A$1" Then
      Application.Help "Notepad.hlp"
   End If
End Sub
```

Перевычисление функции пользователя при изменениях в ячейках рабочего листа

Метод `Volatile` объекта `Application` вызывает перевычисление функции пользователя при изменениях в любых ячейках рабочего листа.

```
Volatile(Volatile)
```

Здесь *`Volatile`* — необязательный параметр, причем, если он принимает значение `True`, подразумеваемое по умолчанию, перевычисление функции пользователя происходит при изменениях в любых ячейках рабочего листа, а если он принимает значение `False`, только при изменении значения в тех ячейках, на которые функция имеет ссылки. Например:

```
Function Power(ByVal x As Double, ByVal d As Double) As Double
   Application.Volatile
   Power = x ^ d
End Function
```

Установка выполнения специфицированной процедуры при нажатии заданной комбинации клавиш

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

```
OnKey(Key, Procedure)
```

- *`Key`* — обязательный параметр, задающий строку, определяющую комбинацию клавиш, которая должна быть нажата. В этой строке можно также указывать специальные клавиши, используя коды, приведенные в табл. 3.9. Допустимо использование сочетания одновременно нажатой специальной клавиши и клавиши модификатора. С этой целью установлены коды клавиш-модификаторов, перечисленные в табл. 3.10;
- *`Procedure`* — необязательный параметр, задающий имя процедуры, выполняемой при нажатии комбинации клавиш, указанных в качестве значения параметра *`Key`*. Если параметр опущен, то выполняется действие, которое в системе зарезервировано за этой комбинацией клавиш.

Таблица 3.9. *Коды специальных клавиш*

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

Таблица 3.10. *Коды клавиш-модификаторов*

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

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

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

Назначение выполнения процедуры на определенное время

Метод OnTime объекта Application назначает выполнение указанной процедуры на определенное время.

```
OnTime(EarliestTime, Procedure, LatestTime, Schedule)
```

- *EarliestTime* — обязательный параметр типа Variant, задающий момент запуска процедуры;
- *Procedure* — обязательный параметр типа String, задающий имя процедуры;
- *LatestTime* — необязательный параметр типа Variant. Если на момент запуска процедуры MS Excel не может ее запустить в силу того, что он выполняет другое действие, то *LatestTime* определяет последнее время ее запуска. Если этот параметр опущен, то MS Excel будет ждать до тех пор, пока не сможет выполнить данную процедуру;
- *Schedule* — необязательный параметр типа Variant. Если его значение равно True, то указанная процедура установлена на выполнение в специфицированное время. Если же значение равно False, то задание снято с исполнения.

Электронные часы в ячейке рабочего листа

Метод OnTime позволяет создать электронные часы. Для этого достаточно рекурсивно вызывать процедуру, в которой считывается текущее время, затем оно выводится в ячейку рабочего листа, после чего найденное время увеличивается на секунду, а уже для вычисленного нового времени устанавливается рекурсивный вызов процедуры самой себя при помощи метода OnTime (листинг 3.35).

Листинг 3.35. Электронные часы в ячейке рабочего листа. Стандартный модуль

```
Sub DemoClock()
   DemoOnTime
End Sub
```

```
Sub DemoOnTime()
   Dim newHour, newMinute, newSecond, newTime
   Cells(1, 1).Value = Now
   newHour = Hour(Now)
   newMinute = Minute(Now)
   newSecond = Second(Now) + 1
   newTime = TimeSerial(newHour, newMinute, newSecond)
   Application.OnTime EarliestTime:=newTime, _
                      Procedure:="DemoOnTime"
End Sub
```

Электронный будильник

Метод `OnTime` обеспечивает простую реализацию будильника. Для этого в качестве значения первого его параметра достаточно указать момент времени, когда будильник должен прозвонить, а в качестве значения второго параметра — ссылку на процедуру, которая и реализует "звон". Например, приводимый ниже код (листинг 3.36) обеспечивает отображение сообщения `Слишком поздно!!!` в 50 минут после полуночи.

Листинг 3.36. Электронный будильник

```
Sub DemoAlarm()
   Application.OnTime EarliestTime:=TimeValue("00:50:00"), _
                      Procedure:="AlarmClock"
End Sub

Sub AlarmClock()
   MsgBox "Слишком поздно!!!"
End Sub
```

Отмена задания у часов

Если значение параметра *`Schedule`* метода `OnTime` принято равным `False`, то установленное при помощи параметров *`EarliestTime`* и *`Procedure`* задание снимается с выполнения. Например, следующий код (листинг 3.37) отменяет запуск процедуры, выполняющей роль будильника из предыдущего раздела, при двойном щелчке на ячейке рабочего листа.

Листинг 3.37. Отмена задания у часов. Модуль рабочего листа

```
Private Sub Worksheet_BeforeDoubleClick(ByVal Target As Range, _
      Cancel As Boolean)
   On Error Resume Next
```

```
   Application.OnTime EarliestTime:=TimeValue("00:50:00"), _
                      Procedure:="AlarmClock", _
                      Schedule:=False
End Sub
```

Закрытие приложения

Метод `Quit` объекта `Application` закрывает приложение. Например, следующий код (листинг 3.38) производит закрытие приложения при двойном щелчке на ячейке рабочего листа без предупреждения пользователя о необходимости сохранить обновленную версию рабочей книги.

Листинг 3.38. Закрытие приложения при двойном щелчке на ячейке рабочего листа. Модуль рабочего листа

```
Private Sub Worksheet_BeforeDoubleClick(ByVal Target As Range, _
   Cancel As Boolean)
   Application.DisplayAlerts = False
   Application.Quit
End Sub
```

Пересечение и объединение диапазонов

Методы `Intersect` и `Union` объекта `Application` находят диапазон, являющийся пересечением или объединением указанных диапазонов.

```
Intersect(Arg1, Arg2, ...)
Union(Arg1, Arg2, ...)
```

Здесь *Arg1*, *Arg2*, ... — параметры, имеющие тип `Range`, специфицирующие пересекаемые или объединяемые диапазоны. Например, в следующем коде в ячейки диапазона, являющегося объединением диапазонов **A1:A3** и **C1:C3**, вводится формула рабочего листа `=RAND()`.

```
Dim bigRange As Range
Set bigRange = Application.Union(Range("A1:A3"), Range("C1:C3"))
bigRange.Formula = "=RAND()"
```

Проверка принадлежности ячейки выделенному диапазону

Метод `Intersect` позволяет легко проверять, принадлежит ли данная ячейка указанному диапазону. Например, следующий код (листинг 3.39) при нажатии правой кнопки мыши проверяет, принадлежит ли ячейка **A1** выделен-

ному диапазону. Метод Intersect находит диапазон, являющийся пересечением ячейки **A1** и выделенного диапазона, возвращаемого параметром Target процедуры обработки события BeforeRightClick объекта Worksheet. Остается только убедиться, не пусто ли это пересечение, т. е. не возвращает ли метод Intersect значение Nothing.

Листинг 3.39. Проверка принадлежности ячейки выделенному диапазону. Модуль рабочего листа

```
Private Sub Worksheet_BeforeRightClick(ByVal Target As Range, _
      Cancel As Boolean)
   Dim rg As Range
   Set rg = Range("A1")
   If Application.Intersect(Target, rg) Is Nothing Then
      MsgBox "Не содержит ячейку A1"
   Else
      MsgBox "Содержит ячейку A1"
   End If
End Sub
```

Запуск макроса или вызов функции

Метод Run объекта Application производит запуск макроса или вызов функции.

```
Run(Macro, Arg1, Arg2, Arg3, Arg4, Arg5, Arg6, Arg7, Arg8, Arg9, Arg10,
↳Arg11, Arg12, Arg13, Arg14, Arg15, Arg16, Arg17, Arg18, Arg19, Arg20,
↳Arg21, Arg22, Arg23, Arg24, Arg25, Arg26, Arg27, Arg28, Arg29, Arg30)
```

- *Macro* — необязательный параметр типа Variant, задающий ссылку на запускаемый макрос. Его значением может быть либо строка, содержащая имя макроса или функции, либо объект типа Range, указывающий местоположение функции, или идентификатор регистра зарегистрированной DLL (XLL) функции;
- *Arg1*—*Arg30* — необязательные параметры типа Variant, задающие значения, которые должны быть переданы в качестве значений параметров функции или макроса.

Метод Run возвращает то значение, которое вычисляется вызываемой функцией или макросом. В качестве примера метода Run приведем следующий код (листинг 3.40), в котором вызывается как макрос, который не возвращает никакого значения, так и функция, которая его возвращает.

Листинг 3.40. Запуск макроса или вызов функции

```
Sub TestRun()
  Application.Run "Test", "Привет"
  Dim s As Double
  s = Application.Run("TestSum", Range("A1:A3"))
  MsgBox s
End Sub

Sub Test(ByVal msg As String)
   MsgBox msg
End Sub

Function TestSum(ByVal rng As Range) As Double
   Dim s As Double
   Dim r As Range
   For Each r In rng.Cells
      s = s + r.Value
   Next
   TestSum = s
End Function
```

Сохранение изменений во всех рабочих книгах

Метод `Save` объекта `Application` сохраняет на диск изменения указанной рабочей книги. Следующий код (листинг 3.41), обрабатывающий событие `BeforeRightClick` объекта `Worksheet`, при щелчке правой кнопкой на ячейке **A1** обеспечивает сохранение всех рабочих книг, открытых в данном приложении с последующим его закрытием.

Листинг 3.41. Сохранение изменений во всех рабочих книгах. Модуль рабочего листа

```
Private Sub Worksheet_BeforeRightClick(ByVal Target As Range, _
      Cancel As Boolean)
   If Target Is Range("A1") Then
      Dim wb As Workbook
      For Each wb In Application.Workbooks
         wb.Save
      Next
      Application.Quit
   End If
End Sub
```

Приостановка работы приложения до указанного времени

Метод `Wait` объекта `Application` приостанавливает работу приложения без остановки работы других программ до указанного момента времени. Возвращает значение `True`, если специфицированный момент времени наступил.

```
Wait(Time)
```

Здесь параметр *`Time`* определяет время, в которое предполагается возобновить работу приложения.

В следующем примере устанавливается возобновление работы приложения на 17 часов:

```
Application.Wait "17:00:00"
```

В данном примере приложение прерывает выполнение до указанного момента времени и начинает работу с отображением сообщения:

```
Dim delay As Date
delay = #11:11:00 PM#
If Application.Wait(delay) Then
   MsgBox " Пора взяться за дело"
End If
```

В последнем коде выполнение приложения задерживается на 10 секунд:

```
Dim delay As Date
delay = Now + TimeValue("00:00:10")
Application.Wait delay
```

События объекта *Application*

Прежде чем применить событие объекта `Application`, необходимо создать модуль класса и там объявить объект типа `Application`, причем при этом объявлении использовать ключевое слово `WithEvents`. Например, как сделано в следующем коде (листинг 3.42, *а*). Кроме того, переименуйте модуль класса из безликого `Class1`, например, в `MyAppWithEvent`.

Листинг 3.42, *а*. Объявление объекта типа `Application`. Модуль класса

```
Public WithEvents App As Application
```

После этого имя объекта `App` будет добавлено в список объектов, приводимый в списке **General** модуля класса. В списке **Declarations** приводятся ассоциированные с ним события. Например, следующий код (листинг 3.42, *б*)

реализует обработку двух событий: `WorkbookActivate` и `WorkbookDeactivate`, генерируемых при активизации и деактивизации рабочей книги.

Листинг 3.42, *б*. Процедуры, обрабатывающие события объекта типа `Application`. Модуль класса

```
Private Sub App_WorkbookActivate(ByVal Wb As Workbook)
   MsgBox "Книга " & Wb.Name & " активизирована"
End Sub

Private Sub App_WorkbookDeactivate(ByVal Wb As Workbook)
   MsgBox "Книга " & Wb.Name & " деактивизирована"
End Sub
```

Теперь, для того чтобы процедура, обрабатывающая событие, исполнялась, остается объявить объект типа `MyAppWithEvent`, например, как это делается в следующим коде (листинг 3.42, *в*).

Листинг 3.42, *в*. Объявление объекта типа `MyAppWithEvent`. Модуль `ЭтаКнига`

```
Dim a As New MyAppWithEvent

Private Sub Workbook_Open()
   Set a.App = Application
End Sub
```

В заключение в табл. 3.11 перечислим события, связанные с объектом `Application`.

Таблица 3.11. *События объекта* `Application`

Событие	Описание
`NewWorkbook`	Создание новой рабочей книги
`SheetActivate`	Активизация листа. Может быть как рабочий лист, так и лист с диаграммой
`SheetBeforeDoubleClick`	Двойной щелчок на рабочем листе
`SheetBeforeRightClick`	Щелчок правой кнопкой мыши на рабочем листе. Событие происходит до того, как генерируется определенное по умолчанию аналогичное событие. Если значение параметра `Cancel` процедуры обработки этого события установить равным `True`, то определенное по умолчанию действие не выполняется

Таблица 3.11 *(окончание)*

Событие	Описание
SheetCalculate	Пересчет на рабочем листе или изменение данных в диаграмме
SheetChange	Изменение данных в ячейках рабочего листа
SheetDeactivate	Деактивизация листа. Может быть как рабочий лист, так и лист с диаграммой
SheetFollowHyperlink	Щелчок на гиперссылке
SheetPivotTableUpdate	Обновление сводной таблицы
SheetSelectionChange	Смена выделения на рабочем листе
WindowActivate	Активизация окна
WindowDeactivate	Деактивизация окна
WindowResize	Изменение размеров окна
WorkbookActivate	Активизация рабочей книги
WorkbookAddinInstall	Инсталляция надстройки
WorkbookAddinUninstall	Удаление надстройки
WorkbookBeforeClose	Перед закрытием рабочей книги. Если значение параметра Cancel процедуры обработки этого события установить равным True, то по завершении ее работы книга не закроется
WorkbookBeforePrint	Перед печатью рабочей книги
WorkbookBeforeSave	Перед сохранением рабочей книги
WorkbookDeactivate	Деактивизация рабочей книги
WorkbookNewSheet	Добавление нового листа в рабочую книгу
WorkbookOpen	Открытие рабочей книги
WorkbookPivotTableCloseConnection	Закрытие связей со сводной таблицей
WorkbookPivotTableOpenConnection	Установка связей со сводной таблицей

Семейство *Workbooks*

Семейство Workbooks инкапсулирует в себе все рабочие листы рабочей книги. В этом семействе имеются два основных свойства (табл. 3.12), возвращающие число элементов семейства и конкретный элемент семейства.

Таблица 3.12. Свойства семейства `Workbooks`

Свойство	Описание
`Count`	Возвращает число элементов семейства
`Item`	Возвращает элемент семейства

Методы семейства *Workbooks*

Методы семейства `Workbooks` позволяют как открывать существующую книгу, так и создавать новую книгу, причем как пустую, так и заполненную на основе данных, либо из текстового файла, либо из XML-файла, либо из базы данных. Также методы семейства `Workbooks` позволяют управлять считыванием данных с сервера. В нижеприведенных разделах на примерах рассмотрены следующие основные методы этого семейства:

`Add` `Open`
`CanCheckOut` `OpenDatabase`
`CheckOut` `OpenText`
`Close` `OpenXML`

Создание новой рабочей книги

Метод `Add` семейства `Workbooks` создает новую рабочую книгу.

`Add(`*`Template`*`)`

Здесь *`Template`* — необязательный параметр, определяющий то, какой будет создана новая книга. В качестве значения это параметра можно указать:

- имя файла, который будет использован в качества шаблона для создаваемой книги;
- одну из следующих констант `XlWBATemplate`: `xlWBATChart`, `xlWBATExcel4IntlMacroSheet`, `xlWBATExcel4MacroSheet` и `xlWBATWorksheet`.

Если этот параметр опущен, то создается рабочая книга с числом листов, заданных свойством `SheetsInNewWorkbook`.

Проверка того, можно ли редактировать книгу, расположенную на сервере

Метод `CanCheckOut` семейства `Workbooks` возвращает логическое значение, говорящее о том, можно ли редактировать книгу, расположенную на сервере.

`CanCheckOut(`*`FileName`*`)`

Здесь *FileName* — обязательный параметр, специфицирующий имя книги. Например, следующий код (листинг 3.43) проверяет, можно ли редактировать специфицированную книгу, расположенную на сервере

Листинг 3.43. Проверка того, можно ли редактировать книгу, расположенную на сервере

```
Sub DemoCanCheckOut()
   Dim doc As String
   doc = "http://localhost/03_15.xls"
   If Workbooks.CanCheckOut(Filename:=doc) Then
      MsgBox "Редактировать разрешено"
   Else
      MsgBox "Редактировать запрещено"
   End If
End Sub
```

Редактирование книги, расположенной на сервере

Метод `CheckOut` семейства `Workbooks` возвращает строку, представляющую собой книгу, расположенную на сервере и предназначенную для редактирования.

```
CheckOut(FileName)
```

Здесь *FileName* — обязательный параметр, указывающий имя редактируемой книги. Например, в следующем коде (листинг 3.44) сначала проверяется допустимость редактирования книги и только затем производится ее редактирование.

Листинг 3.44. Редактирование книги, расположенной на сервере

```
Dim doc As String
doc = "http://localhost/03_15.xls"
If Workbooks.CanCheckOut(Filename:=doc) Then
   MsgBox "Редактировать разрешено"
   Workbooks.CheckOut doc
Else
   MsgBox "Редактировать запрещено"
End If
```

Закрытие рабочей книги с сохранением и не с сохранением изменений

Метод `Close` семейства `Workbooks` позволяет закрывать рабочую книгу как с сохранением, так и не с сохранением изменений.

```
Close (SaveChanges, FileName, RouteWorkbook)
```

- *SaveChanges* — необязательный параметр. Если в книге не было изменений, то данный параметр игнорируется. Если изменения были, то при его значении равным True, они сохраняются, а при значении False — не сохраняются. При отсутствии параметра отображается окно, требующее от пользователя подтверждения на сохранение изменений;
- *FileName* — необязательный параметр, задающий имя файла, в который сохраняются изменения в книге;
- *RouteWorkbook* — необязательный параметр, устанавливающий необходимость в отсылке книги следующему пользователю.

Например, следующая процедура, обрабатывающая событие BeforeDoubleClick объекта Worksheet, обеспечивает закрытие активной рабочей книги без сохранения изменений при двойном щелчке на ячейке **A1** и с сохранением изменений при двойном щелчке на ячейке **B1** (листинг 3.45).

Листинг 3.45. Закрытие книги. Модуль рабочего листа

```
Private Sub Worksheet_BeforeDoubleClick(ByVal Target As Range, _
      Cancel As Boolean)
   If Target.Address = "$A$1" Then
      ActiveWorkbook.Close SaveChanges:=False
   ElseIf Target.Address = "$B$1" Then
      ActiveWorkbook.Close SaveChanges:=True
   End If
End Sub
```

Открытие рабочей книги

Метод Open семейства Workbooks открывает указанную рабочую книгу.

```
Open(FileName, UpdateLinks, ReadOnly, Format, Password, WriteResPassword,
↳IgnoreReadOnlyRecommended, Origin, Delimiter, Editable, Notify,
↳Converter, AddToMru, Local, CorruptLoad, OpenConflictDocument)
```

- *FileName* — обязательный параметр, специфицирующий имя файла;
- *UpdateLinks* — необязательный параметр, указывающий на то, как должны обновляться связи в рабочей книге;
- *ReadOnly* — необязательный параметр. Если он принимает значение True, то книга открывается в режиме "только для чтения";
- *Format* — необязательный параметр, специфицирующий разделитель, если MS Excel открывает текстовый файл;
- *Password* — необязательный параметр, указывающий пароль для открытия защищенной рабочей книги;

- *WriteResPassword* — необязательный параметр, специфицирующий пароль для внесения изменений в защищенную от записи книгу;
- *IgnoreReadOnlyRecommended* — необязательный параметр, указывающий, надо ли отображать предупреждение о том, что книга была сохранена в режиме "только для чтения".
- *Origin* — необязательный параметр, специфицирующий, какой текстовый файл создан, если MS Excel открывает текстовый файл;
- *Delimiter* — необязательный параметр, указывающий пользовательский разделитель. Этот параметр используется только в случае, если значение параметра *Format* равно 6;
- *Editable* — необязательный параметр, позволяющий открыть данный шаблон для редактирования;
- *Notify* — необязательный параметр, который при значении `False` запрещает любые попытки открытия недостижимых по каким-либо причинам файлов;
- *Converter* — необязательный параметр, специфицирующий конвертер, если таковой требуется при открытии файла;
- *AddToMru* — необязательный параметр, значение которого `True` обеспечивает добавление файла в список недавно использованных файлов;
- *Local* — необязательный параметр, специфицирующий, надо ли сохранять файл в соответствии с локальными установками MS Excel;
- *CorruptLoad* — необязательный параметр, указывающий, открывается нормальный или восстановленный файл;
- *OpenConflictDocument* — необязательный параметр, позволяющий открыть конфликтные документы.

Например, следующая инструкция открывает рабочую книгу Report.xls.

```
Workbooks.Open "Report.xls"
```

Открытие текстового файла как новой рабочей книги

Метод `OpenText` семейства `Workbooks` открывает текстовый файл как новую рабочую книгу.

```
OpenText(FileName, Origin, StartRow, DataType, TextQualifier,
↳ConsecutiveDelimiter, Tab, Semicolon, Comma, Space, Other, OtherChar,
↳FieldInfo, TextVisualLayout, DecimalSeparator, ThousandsSeparator,
↳TrailingMinusNumbers, Local)
```

У этого метода есть много параметров, причем все они, за исключением параметра *FileName*, специфицирующего имя текстового файла, являются

необязательными. Смысл этих параметров очевиден из их имен. Разберем в качестве примера инструкцию, которая открывает текстовый файл Report.txt.

```
Страна         Объем продаж
Германия        1231233
Италия          34554645
Франция         23432654
Япония          7654534
```

В этой инструкции параметр *FileName* метода OpenText задает имя файла, параметр *Origin* определяет, что файл создан на основе Windows-кодировки, параметр *StartRow* задает строку, с которой информация считывается из текстового файла, параметр *DataType* определяет, имеют ли столбцы постоянную или переменную ширину (как в данном случае), а параметр *Tab* указывает на то, что разделителем между полями в текстовом файле является знак табуляции.

```
Workbooks.OpenText Filename:="C:\Report.txt", Origin:=xlWindows, _
                   StartRow:=1, DataType:=xlDelimited, Tab:=True
```

Данный же тексовый файл (Test.txt)

```
"Иванов","Иван",5/1/73
"Петров","Петр",1/5/81
"Сидоров","Сидор",3/11/75
```

можно открыть при помощи следующей инструкции:

```
Workbooks.OpenText Filename:="C:\Test.txt", Origin:=xlWindows, _
                   StartRow:=1, DataType:=xlDelimited, _
                   TextQualifier:=xlTextQualifierDoubleQuote, _
                   ConsecutiveDelimiter:=True, _
                   Comma:=True, _
                   FieldInfo:=Array(Array(1, xlTextFormat), _
                          Array(2, xlTextFormat), Array(3, xlMDYFormat))
```

Открытие базы данных

Метод OpenDatabase семейства Workbooks открывает указанную базу данных.

```
OpenDatabase(FileName, CommandText, CommandType, BackgroundQuery,
↳ImportDataAs)
```

- *FileName* — обязательный параметр, содержащий ссылку на открываемую базу данных;
- *CommandText* — необязательный параметр, задающий запрос;

- *CommandType* — необязательный параметр, определяющий тип запроса;
- *BackgroundQuery* — необязательный параметр, описывающий параметры запроса;
- *ImportDataAs* — необязательный параметр, указывающий тип запроса.

Например, следующая инструкция создает новую рабочую книгу, в которую из учебной базы Борей.mdb корпорации Microsoft считывает таблицу **Сотрудники**, и вставляет данные из нее в рабочий лист.

```
Workbooks.OpenDatabase Filename:="C:\Борей.mdb", _
                       CommandText:="Сотрудники", _
                       CommandType:=xlCmdTable
```

Открытие XML-файла

Метод `OpenXML` семейства `Workbooks` открывает XML-файл.

```
OpenXML(FileName, Stylesheets)
```

- *FileName* — обязательный параметр, содержащий ссылку на открываемый XML-файл;
- *Stylesheets* — необязательный параметр типа `Variant`, состоящий либо из одного значения, либо из массива значений, указывающих на используемые XSLT таблицы стилей.

Например, следующая инструкция открывает файл Catalog.xml и считывает его содержимое в новую рабочую книгу.

```
Workbooks.OpenXML Filename:="C:\Catalog.xml"
```

Объект *Workbook*

В иерархии MS Excel объект `Workbook` идет сразу после объекта `Application` и представляет файл рабочей книги. Рабочая книга хранится в файлах формата либо XLS (стандартная рабочая книга), либо XLA (полностью откомпилированное приложение). Объект `Workbook` возвращается либо как элемент семейства `Workbooks` соответствующим свойством, либо как активная рабочая книга — свойством `ActiveWorkbook`, либо как книга, в которой данный код выполняется свойством `ThisWorkbook`.

Свойства объекта *Workbook*

Объект `Workbook` обладает большим набором свойств. Свойства `ActiveChart` и `ActiveSheet` возвращают активную диаграмму и лист. Свойства `Charts`, `Comments`, `Names`, `Sheets`, `Windows`, `Worksheets` возвращают семейства диаграмм,

примечаний, имен, листов, окон и рабочих листов данной книги. Перечислим основные свойства объекта `Workbook` (табл. 3.13).

Таблица 3.13. *Основные свойства объекта* `Workbook`

Свойство	Описание
`ActiveChart`	Возвращает активную диаграмму
`ActiveSheet`	Возвращает активный лист книги. В следующем примере устанавливается имя активного рабочего листа `ActiveSheet.Name = "Отчет"`
`Author`	Устанавливает имя автора книги
`AutoUpdateFrequency`	Задает частоту обновления книги, совместно используемой несколькими пользователями
`AutoUpdateSaveChanges`	Указывает, надо ли автоматически пересылать обновления другому пользователю при их сохранении и совместном использовании книги
`CalculationVersion`	Возвращает версию MS Excel, в которой рабочая книга в последний раз была полностью пересчитана
`ChangeHistoryDuration`	Задает число дней, в течение которых запоминаются изменения в книге при ее совместном использовании
`Charts`	Возвращает семейство всех диаграмм книги (которые не внедрены в рабочие листы)
`CommandBars`	Возвращает семейство всех панелей управления
`Comments`	Возвращает семейство всех примечаний
`CreateBackup`	Устанавливает, надо ли создавать backup-копию файла при сохранении изменений в книге
`DisplayDrawingObjects`	Управляет отображением графических объектов. Допустимыми значениями являются следующие константы `XlDisplayDrawingObjects`: `xlDisplayShapes`, `xlPlaceholders` и `xlHide`
`EnableAutoRecover`	Позволяет автоматически восстанавливать поврежденную книгу

Таблица 3.13 *(продолжение)*

Свойство	Описание
`FileFormat`	Задает формат файла рабочей книги
`FullName`	Задает полное имя рабочей книги
`FullNameURLEncoded`	Задает имя объекта совместо с указанием пути, где расположен файл
`HTMLProject`	Возвращает объект `HTMLProject`, представляющий HTML-проект книги в MS Script Editor
`HasPassword`	Возвращает значение `True`, если книга защищена паролем, и значение `False` — в противном случае
`HasRoutingSlip`	Проверяет, ассоциирована ли с книгой маршрутизация ее рассылки пользователям
`HighlightChangesOnScreen`	Выделяет на экране изменения при совместной работе с книгой нескольких пользователей
`IsAddin`	Проверяет, используется ли книга как надстройка
`KeepChangeHistory`	Проверяет, запоминаются ли исправления, произведенные пользователями при совместной работе над книгой
`ListChangesOnNewSheet`	Если значение свойства равно `True`, то исправления, произведенные пользователями при совместной работе над книгой, отображаются в новом рабочем листе
`MultiUserEditing`	Если значение свойства равно `True`, то открытая книга разрешена для совместного использования несколькими пользователями
`Name`	Задает имя книги
`Names`	Задает семейство всех имен, использованных в книге
`Password`	Задает пароль для защиты книги
`PasswordEncryptionAlgorithm`	Задает алгоритм шифрования, используемый MS Excel для кодировки пароля
`PasswordEncryptionFileProperties`	Проверяет, шифруются ли свойства документа

Таблица 3.13 *(продолжение)*

Свойство	Описание
PasswordEncryptionKeyLength	Задает стойкость ключа
PasswordEncryptionProvider	Задает тип шифрования, используемого MS Excel для кодировки пароля
Path	Возвращает путь к книге
PrecisionAsDisplayed	Задает, надо ли производить вычисления с данными с той точностью, как они отображаются на рабочем листе
ProtectStructure	Проверяет, защищена ли структура рабочей книги
ProtectWindows	Проверяет, защищены ли окна рабочей книги
PublishObjects	Возвращает семейство PublishObjects объектов PublishObject, каждый из которых представляет элемент рабочей книги, сохраненный для использования в Web-странице
ReadOnly	Проверяет, открыта ли книга в режиме "только для чтения"
RevisionNumber	Возвращает количество сохранений книги, пока она была открыта для совместного использования
Routed	Проверяет, была ли книга отправлена по маршруту следующему пользователю
RoutingSlip	Возвращает объект RoutingSlip, который инкапсулирует в себе данные, необходимые при маршрутизированной рассылке книги
SaveLinkValues	Сохранение внешних связей вместе с книгой
Saved	Возвращает значение True, если никаких изменений не было сделано в книге с момента ее последнего сохранения, и значение False — в противном случае
Sheets	Возвращает семейство всех листов книги
ShowPivotTableFieldList	Проверяет, можно ли отобразить список полей сводной таблицы
SmartTagOptions	Возвращает объект SmartTagOptions, инкапсулирующий в себе свойства смарт-тега

Таблица 3.13 *(окончание)*

Свойство	Описание
`Styles`	Возвращает семейство всех стилей, используемых в книге
`Title`	Задает заголовок Web-страницы, когда книга будет сохранена как Web-страница
`UpdateLinks`	Указывает на то, как обновляются связи со вложенными в книгу OLE-объектами. Допустимы следующие константы `XlUpdateLinks`: `xlUpdateLinksAlways`, `xlUpdateLinksNever`, `xlUpdateLinksUserSetting`
`UserControl`	Возвращает значение `True`, если книга была создана пользователем, и значение `False`, если она была сконструирована программно
`WebOptions`	Возвращает объект `WebOptions`, который инкапсулирует в себе данные, используемые MS Excel при сохранении книги как Web-страницы
`Windows`	Возвращает семейство окон книги
`Worksheets`	Возвращает семейство всех рабочих листов книги
`WritePassword`	Устанавливает пароль для записи книги
`WriteReserved`	Проверяет, защищена ли книга от записи
`WriteReservedBy`	Задает имя пользователя, у которого в данный момент имеются полномочия на защиту книги от записи

Закрытие книги без сохранения изменений

Свойство `Saved` объекта `Workbook` возвращает значение `True`, если никаких изменений не было сделано в книге с момента ее последнего сохранения, и значение `False` — в противном случае. Поэтому если в процедуру обработки события `BeforeClose` объекта `Workbook` добавить инструкцию

```
ActiveWorkbook.Saved = True
```

то книга будет сохранена без сохранения изменений, внесенных в нее с момента последнего ее сохранения (листинг 3.46).

Листинг 3.46. Закрытие книги без сохранения изменений. Модуль ЭтаКнига

```
Private Sub Workbook_BeforeClose(Cancel As Boolean)
   ActiveWorkbook.Saved = True
End Sub
```

Маршрутизация рассылки книги

Метод `Route` осуществляет рассылку книги по получателям, перечисленным в свойстве `Recipients` объекта `RoutingSlip`. Следующий код (листинг 3.47) демонстрирует рассылку активной рабочей книги двум получателям.

Листинг 3.47. Маршрутизация рассылки книги

```
Sub DemoRoute()
   ActiveWorkbook.HasRoutingSlip = True
   With ActiveWorkbook.RoutingSlip
      .Delivery = xlOneAfterAnother
      .Recipients = Array("James Bond", "Crazy Max")
      .Subject = "Это " & ActiveWorkbook.Name
      .Message = "Что вы думаете о ней"
   End With
   ActiveWorkbook.Route
End Sub
```

Методы объекта *Workbook*

Объект `Workbook` имеет обширную коллекцию методов, позволяющих реализовывать большой спектр операций от активизации книги до составления сложного отчета с помощью сводной таблицы. В табл. 3.14 перечислены основные методы объекта `Workbook`.

Таблица 3.14. *Основные методы объекта* `Workbook`

Метод	Описание
`AcceptAllChanges`	Принимает все изменения в совместно используемой книге
`Activate`	Активизирует рабочую книгу так, что ее первый рабочий лист становится активным
`AddToFavorites`	Добавляет книгу в папку **Избранное**
`BreakLink`	Конвертирует формулы или OLE-объекты между связанными книгами в значения

Таблица 3.14 *(продолжение)*

Метод	Описание
`CanCheckIn`	Проверяет и устанавливает, можно ли редактировать книгу, расположенную на сервере
`ChangeFileAccess`	Изменяет режим доступа к книге
`ChangeLink`	Изменяет тип связи, установленный между документами
`CheckIn`	Производит редактирование книги, расположенной на сервере
`Close`	Закрывает книгу
`DeleteNumberFormat`	Удаляет пользовательский числовой формат
`EndReview`	Прекращает редактирование файла, который был отослан методом `SendForReview`
`ExclusiveAccess`	Устанавливает для текущего пользователя эксклюзивный доступ к книге
`FollowHyperlink`	Производит переход по гиперссылке
`NewWindow`	Открывает новое окно в книге
`PivotCaches`	Возвращает семейство `PivotCaches`, представляющее собой кэш-таблицу, отведенную под сводную таблицу
`PivotTableWizard`	Создает сводную таблицу
`Post`	Пересылает книгу в папку общего доступа. Метод работает только с клиентами MS Exchange, соединенными с MS Exchange-сервером
`PrintOut`	Печатает книги
`PrintPreview`	Задает предварительный просмотр книги
`Protect`	Защищает рабочую книгу от внесения в нее изменений
`ProtectSharing`	Защищает книгу при совместном использовании
`PurgeChangeHistoryNow`	Удаляет данные старше указанного срока из журнала изменений
`RefreshAll`	Обновляет все внешние данные и отчеты, созданные на основе сводной таблицы
`RejectAllChanges`	Отменяет все изменения в книге при совместном ее использовании
`ReloadAs`	Перегружает книгу, созданную на основе HTML-документа, используя указанную кодировку
`RemoveUser`	Отключает указанного пользователя от книги при ее совместном использовании

Таблица 3.14 (окончание)

Метод	Описание
ReplyWithChanges	Отсылает сообщения, подтверждающие завершение редактирования книги
Save	Сохраняет книгу
SaveAs	Сохраняет книгу в другой файл
SaveCopyAs	Сохраняет копию книги
SendMail	Отсылает почту, используя встроенные средства MS Mail
Unprotect	Снимает защиту с рабочей книги
UnprotectSharing	Снимает защиту с совместно используемой книги
UpdateFromFile	Обновляет из файла книги с атрибутом "только для чтения"
UpdateLink	Обновляет связи MS Excel и OLE
WebPagePreview	Задает предварительный просмотр книги, прежде чем она будет сохранена как Web-страница

Переход к документу

Метод FollowHyperlink объекта Workbook производит переход к уже открытому документу или загрузке указанного документа с последующим его отражением в подходящем приложении.

```
FollowHyperlink(Address, SubAddress, NewWindow, AddHistory,
↳ExtraInfo, Method, HeaderInfo)
```

- *Address* — обязательный параметр, указывающий адрес целевого документа;
- *SubAddress* — необязательный параметр, указывающий относительное местоположение по отношению к целевому документу;
- *NewWindow* — необязательный параметр. Если он принимает значение True, документ отображается в новом окне, если же его значение равно False, то в том же самом;
- *AddHistory* — не используется, зарезервирован для будущих версий MS Excel;
- *ExtraInfo* — необязательный параметр, содержащий дополнительную информацию, используемую для HTTP-протокола;
- *Method* — необязательный параметр, специфицирующий то, как передается информация из параметра *ExtraInfo*;

- *HeaderInfo* — необязательный параметр, содержащий заголовочную информацию, передаваемую при HTTP-запросе.

Например, следующая процедура обработки события BeforeDoubleClick объекта Worksheet обеспечивает загрузку книги Test.xls при двойном щелчке на ячейке **A1** и отображение Web-страницы **http://www.microsoft.com** при двойном щелчке на ячейке **B1** (листинг 3.48).

Листинг 3.48. Переход к документу. Модуль рабочего листа

```
Private Sub Worksheet_BeforeDoubleClick(ByVal Target As Range, _
                                        Cancel As Boolean)
   If Target.Address = "$A$1" Then
      ActiveWorkbook.FollowHyperlink Address:="Test.xls", _
                                     NewWindow:=True
   ElseIf Target.Address = "$B$1" Then
      ActiveWorkbook.FollowHyperlink Address:="http://www/.microsoft.com", _
                                     NewWindow:=True
  End If
End Sub
```

Установка и снятие защиты книги

Защита на рабочую книгу от внесения в нее изменений устанавливается методом Protect, а снимается — методом Unprotect.

```
Protect(Password, Structure, Windows)
```

- *Password* — необязательный параметр, задающий пароль для защиты книги. Если параметр опущен, то книга защищена без пароля;
- *Structure* — необязательный параметр, устанавливающий, защищена ли структура книги, т. е. взаимное расположение листов;
- *Windows* — необязательный параметр, определяющий, защищены ли окна книги.

Защита с рабочей книги снимается методом Unprotect.

```
Unprotect(Password)
```

Здесь *Password* — необязательный параметр, указывающий пароль защиты.

В приводимом ниже коде (листинг 3.49) обрабатывается событие BeforeRightClick объекта Worksheet. При щелчке правой кнопкой на ячейке **A1** устанавливается защита структуры и окон рабочей книги. Щелчок правой же кнопкой на ячейке **B1** снимает эту защиту.

Листинг 3.49. Установка и снятие защиты книги. Модуль рабочего листа

```
Private Sub Worksheet_BeforeRightClick(ByVal Target As Range, _
                                       Cancel As Boolean)
   If Target.Address = "$A$1" Then
      Cancel = True
      ThisWorkbook.Protect Password:="wow", Structure:=True, _
                           Windows:=True
   End If
   If Target.Address = "$B$1" Then
      Cancel = True
      ThisWorkbook.Unprotect Password:="wow"
   End If
End Sub
```

События объекта *Workbook*

У объекта `Workbook` имеется большая коллекция событий, перечисленных в табл. 3.15. Они позволяют управлять процессом открытия книги, печатью, отслеживать любые изменения, произведенные в ней, и т. д.

Таблица 3.15. *События объекта* `Workbook`

Событие	Описание
`Activate`	Генерируется при активизации рабочей книги
`AddinInstall`	Генерируется, когда рабочая книга инсталлируется как надстройка
`AddinUninstall`	Генерируется, когда рабочая книга удаляется как инсталлированная надстройка
`BeforeClose`	Генерируется перед закрытием рабочей книги
`BeforePrint`	Генерируется перед печатью
`BeforeSave`	Генерируется перед сохранением рабочей книги
`Deactivate`	Генерируется при потере фокуса рабочей книгой
`NewSheet`	Генерируется при создании нового листа
`Open`	Генерируется при открытии рабочей книги
`PivotTableCloseConnection`	Генерируется при закрытии связи сводной таблицы с источником данных
`PivotTableOpenConnection`	Генерируется при установке связи сводной таблицы с источником данных

Таблица 3.15 *(окончание)*

Событие	Описание
SheetActivate	Генерируется при активизации любого рабочего листа
SheetBeforeDoubleClick	Генерируется при двойном щелчке, но перед выполнением действий, назначенных по умолчанию двойному щелчку
SheetBeforeRightClick	Генерируется при щелчке правой кнопкой мыши, но перед выполнением действий, назначенных по умолчанию этому щелчку
SheetCalculate	Генерируется после завершения всех вычислений во всех рабочих листах и перерисовки диаграмм
SheetChange	Генерируется при изменении значения на любом листе в какой-либо ячейке
SheetDeactivate	Генерируется при потере листом фокуса
SheetFollowHyperlink	Генерируется при щелчке на гиперссылке
SheetPivotTableUpdate	Генерируется после обновления сводной таблицы
SheetSelectionChange	Генерируется при изменении выделения на любом листе
WindowActivate	Генерируется при активизации окна
WindowDeactivate	Генерируется при деактивизации окна
WindowResize	Генерируется при изменении размеров окна

Запрет закрытия рабочей книги

У процедуры обработки события `BeforeClose` объекта `Workbook`, генерируемого перед закрытием рабочей книги, имеется параметр `Cancel`. При значении `True` он блокирует закрытие книги. При значении `False` — закрытие разрешено. Например, следующий код (листинг 3.50) позволяет пользователю закрыть книгу только в случае, если в ячейку **A1** введена строка `Go`.

Листинг 3.50. Запрет закрытия рабочей книги. Модуль `ЭтаКнига`

```
Private Sub Workbook_BeforeClose(Cancel As Boolean)
   If Range("A1").Value = "Go" Then
      Cancel = False
   Else
      Cancel = True
   End If
End Sub
```

Обязательный пересчет данных в рабочей книге перед ее печатью

Процедура обработки события `BeforePrint` объекта `Workbook`, генерируемого перед печатью книги, имеет параметр `Cancel`, позволяющий управлять печатью. Кроме того, если перед печатью надо выполнить какие-либо действия, например произвести обязательный пересчет всех данных, то этот код (листинг 3.51) также должен помещаться в процедуру обработки события `BeforePrint`.

Листинг 3.51. Обязательный пересчет данных в рабочей книге перед ее печатью. Модуль ЭтаКнига

```
Private Sub Workbook_BeforePrint(Cancel As Boolean)
   Dim ws As Worksheet
   For Each ws In Worksheets
      ws.Calculate
   Next
End Sub
```

Обязательный запрос на подтверждение сохранения рабочей книги

Процедура обработки события `BeforeSave` объекта `Workbook`, генерируемого перед сохранением книги, имеет два параметра.

```
Private Sub Workbook_BeforeSave(ByVal SaveAsUi As Boolean, _
                                Cancel As Boolean)
```

- *SaveAsUi* — если принимает значение `True`, отображается окно **Сохранить как**;
- *Cancel* — если значение этого параметра равно `False`, то по завершении процедуры не происходит сохранения книги.

Например, следующий код (листинг 3.52) при попытке сохранения книги обеспечивает отображения окна, требующего от пользователя подтверждения. Сохранение производится только после получения этого подтверждения.

Листинг 3.52. Обязательный запрос на подтверждение сохранения рабочей книги. Модуль ЭтаКнига

```
Private Sub Workbook_BeforeSave(ByVal SaveAsUI As Boolean, _
        Cancel As Boolean)
```

```
   Dim msg As Integer
   msg = MsgBox("Сохранить книгу?", vbYesNo)
   Select Case msg
      Case vbNo
        Cancel = True
      Case vbYes
        Cancel = False
   End Select
End Sub
```

Управление процессом создания нового листа

Процедура обработки события `NewSheet` объекта `Workbook`, генерируемого при создании нового листа, позволяет управлять процессом его создания благодаря параметру *Sh*, который и возвращает ссылку на этот лист. Например, в следующем коде (листинг 3.53) вновь созданному листу присваивается имя, совпадающее с его номером, а сам лист добавляется в конец книги, а не в начало, как происходит по умолчанию.

Листинг 3.53. Управление процессом создания нового листа. Модуль ЭтаКнига

```
Private Sub Workbook_NewSheet(ByVal Sh As Object)
   Sh.Name = Sheets.Count
   Sh.Move After:=Sheets(Sheets.Count — 1)
End Sub
```

Семейство *Worksheets*

Семейство `Worksheets` инкапсулирует в себе все рабочие листы рабочей книги. Основные свойства этого семейства возвращают число элементов и конкретный элемент семейства. Методы же семейства `Worksheets` предоставляют более широкий ассортимент операций, начиная от создания нового листа и заканчивая его удалением. Свойства и методы семейства `Worksheets` перечислены в табл. 3.16 и 3.17.

Таблица 3.16. *Свойства семейства* `Worksheets`

Свойство	Описание
`Count`	Возвращает число элементов семейства
`Item`	Возвращает элемент семейства

Таблица 3.17. *Методы семейства* `Worksheets`

Метод	Описание
`Add`	Создает новый лист. `Add (`*`Before, After, Count, Type`*`)` ♦ *`Before`* — необязательный параметр, специфицирующий лист, перед которым будет добавлен новый; ♦ *`After`* — необязательный параметр, специфицирующий лист, после которого будет добавлен новый; ♦ *`Count`* — необязательный параметр, указывающий число добавляемых листов. По умолчанию его значение равно единице; ♦ *`Type`* — необязательный параметр, задающий тип листа. Его допустимым значением может быть одна из следующих констант `XlSheetType`: `xlWorksheet`, `xlChart`, `xlExcel4MacroSheet` и `xlExcel4IntlMacroSheet`. По умолчанию подразумевается значение `xlWorksheet`
`Copy`	Копирует лист. `Copy (`*`Before, After`*`)` ♦ *`Before`* — необязательный параметр, специфицирующий лист, перед которым будет вставлен данный; ♦ *`After`* — необязательный параметр, специфицирующий лист, после которого будет вставлен данный
`Delete`	Удаляет лист
`Move`	Перемещает лист. `Move (Before, After)` ♦ *`Before`* — необязательный параметр, специфицирующий лист, перед которым будет вставлен данный; ♦ *`After`* — необязательный параметр, специфицирующий лист, после которого будет вставлен данный
`PrintOut`	Печать листа или группы листов. `PrintOut (`*`From, To, Copies, Preview,`* ↳*`ActivePrinter, PrintToFile, Collate,`* ↳*`PrToFileName`*`)` ♦ *`From`* — необязательный параметр, специфицирующий номер листа, с которого начинается печать группы специфицированных рабочих листов; ♦ *`To`* — необязательный параметр, специфицирующий номер последнего листа из группы печатаемых листов;

Таблица 3.17 *(окончание)*

Метод	Описание
(прод.)	♦ *Copies* — необязательный параметр, задающий число печатаемых копий; ♦ *Preview* — необязательный параметр, причем, если его значение равно `True`, то перед осуществлением печати MS Excel отображает окно предварительного просмотра, а если `False` — то печать происходит непосредственно без отображения этого окна; ♦ *ActivePrinter* — необязательный параметр, задающий имя активного принтера; ♦ *PrintToFile* — необязательный параметр, причем, если его значение равно `True`, то печать производится в файл; ♦ *Collate* — необязательный параметр, причем, если его значение равно `True`, MS Excel разбирает документ по печатаемым копиям; ♦ *PrToFileName* — необязательный параметр, указывающий имя выходного файла, если значение параметра *PrintToFile* установлено равным `True`
`PrintPreview`	Отображение окна предварительного просмотра
`Select`	Выбирает специфицированный лист
`FillAcrossSheets`	Копирует указанный диапазон в другие рабочие листы данной коллекции. `FillAcrossSheets (Range, Type)` ♦ *Range* — обязательный параметр, дающий ссылку на копируемый диапазон; ♦ *Type* — необязательный параметр, специфицирующий то, как диапазон будет копироваться. Допустимые значения определяются константами `XlFillWith`: `xlFillWithAll`, `xlFillWithContents` и `xlFillWithFormats`

Объект *Worksheet*

В иерархии MS Excel объект `Worksheet` идет сразу после объекта `Workbook` и представляет собой рабочий лист. Объект `Worksheet` возвращается либо как элемент семейства `Worksheets`, либо как элемент семейства `Sheets` одноименным свойством объекта `Workbook`, либо как активный рабочий лист — свойством `ActiveSheet`.

Свойства объекта *Worksheet*

Объект `Worksheet` обладает огромной коллекцией свойств, предоставляющих разработчику обширные средства от получения доступа к конкретной ячейке до сложной обработки данных типа "фильтрация". В табл. 3.18 перечислены свойства этого объекта.

Таблица 3.18. *Свойства объекта* `Worksheet`

Свойство	Описание
`AutoFilter`	Возвращает объект `AutoFilter`, инкапсулирующий в себе данные об используемом автофильтре для обработки данных, и значение `Nothing`, если фильтр не применяется
`AutoFilterMode`	Возвращает значение `True`, если в ячейках отображаются раскрывающиеся списки автофильтра
`Cells`	Возвращает либо одну ячейку, либо семейство ячеек
`CircularReference`	Возвращает ссылку на диапазон, содержащий циклическую ссылку
`Columns`	Возвращает семейство столбцов
`Comments`	Возвращает семейство `Comments` всех примечаний
`ConsolidationFunction`	Возвращает код функции консолидации данных, используемой при построении итоговой таблицы
`ConsolidationOptions`	Возвращает трехмерный массив, содержащий в себе данные о конструируемой итоговой таблице
`ConsolidationSources`	Возвращает трехмерный массив, содержащий в себе информацию о расположении консолидируемых данных
`DisplayPageBreaks`	Управляет отображением линий разрыва страницы
`EnableAutoFilter`	Если значение этого свойства равно `True`, то раскрывающиеся списки автофильтра достижимы для пользователя даже в случае установки защиты на рабочем листе. При этом должен быть только включен режим защиты пользовательского интерфейса
`EnableCalculation`	Если значение этого свойства равно `False`, то не происходит пересчета формул на рабочем листе, если же его значении равно `True`, то пересчет производится автоматически
`EnableOutlining`	Если значение этого свойства равно `True`, то управление структурой достижимо для пользователя даже в случае установки защиты на рабочем листе. При этом должен быть только включен режим защиты пользовательского интерфейса

Таблица 3.18 *(продолжение)*

Свойство	Описание
`EnablePivotTable`	Если значение этого свойства равно `True`, то управление сводной таблицей достижимо для пользователя даже в случае установки защиты на рабочем листе. При этом должен быть только включен режим защиты пользовательского интерфейса
`EnableSelection`	Определяет, какие ячейки могут быть выбраны на рабочем листе, если на нем установлена защита. Допустимыми значениями являются следующие константы `XlEnableSelection`: `xlNoSelection` (выбор запрещен), `xlNoRestrictions` (выбирать можно все ячейки), `xlUnlockedCells` (выбирать можно только те ячейки, у которых значение свойства `Locked` установлено равным `False`)
`FilterMode`	Это свойство отбражает значение `True`, если на рабочем листе имеются списки, к которым были применены фильтры, в результате чего некоторые их элементы оказались скрытыми
`HPageBreaks`	Возвращает семейство `HpageBreaks`, состоящее из объектов `HPageBreak`, инкапсулирующих в себе информацию о горизонтальных разрывах страницы
`Hyperlinks`	Возвращает семейство `Hyperlinks`, состоящее из объектов `Hyperlink`, инкапсулирующих в себе информацию о гиперссылках
`Index`	Возвращает значения индекса рабочего листа в семействе `Sheets`
`MailEnvelope`	Заголовок отсылаемого по электронной почте сообщения, созданного на основе рабочего листа
`Name`	Имя рабочего листа
`Names`	Возвращает семейство `Names` всех имен данного рабочего листа
`Outline`	Возвращает объект `Outline`, инкапсулирующий в себе информацию о структуре, созданной на рабочем листе
`PageSetup`	Возвращает объект `PageSetup`, инкапсулирующий в себе информацию о параметрах страницы
`ProtectContents`	Свойство "только для чтения", проверяющее, установлена ли защита на содержимое рабочего листа
`ProtectDrawingObjects`	Свойство "только для чтения", проверяющее, установлена ли защита на графические объекты

Таблица 3.18 *(продолжение)*

Свойство	Описание
`ProtectScenarios`	Свойство "только для чтения", проверяющее, установлена ли защита на сценарии
`Protection`	Возвращает объект `Protection`, инкапсулирующий в себе информацию об установленной на листе защите
`ProtectionMode`	Свойство "только для чтения", проверяющее, установлена ли защита только пользовательского интерфейса
`QueryTables`	Возвращает семейство `QueryTables`, состоящее из объектов `QueryTable`, инкапсулирующих в себе информацию о таблицах, построенных на основе запроса к внешним базам данных
`Range`	Возвращает объект `Range`, представляющий диапазон
`Rows`	Возвращает семейство строк
`Scripts`	Возвращает семейство `Scripts`, состоящее из объектов `Script`, инкапсулирующих в себе информацию об используемых скриптах, когда лист сохраняется как Web-страница
`ScrollArea`	Задает диапазон, в котором разрешена прокрутка
`Shapes`	Возвращает семейство `Shapes`, состоящее из объектов `Shape`, инкапсулирующих в себе информацию о расположенных на листе фигурах
`SmartTags`	Возвращает семейство `SmartTags`, состоящее из объектов `SmartTag`, инкапсулирующих в себе информацию о смарт-тегах
`StandardHeight`	Возвращает стандартную высоту ячеек
`StandardWidth`	Возвращает стандартную ширину ячеек
`Tab`	Возвращает объект `Tab`, инкапсулирующий в себе информацию о теге листа
`Type`	Определяет тип листа. Допустимыми значениями могут быть следующие константы `XlSheetType`: `xlChart`, `xlDialogSheet`, `xlExcel4IntlMacroSheet`, `xlExcel4MacroSheet` и `xlWorksheet`
`UsedRange`	Возвращает объект `Range`, который содержит данные
`VPageBreaks`	Возвращает семейство `VPageBreaks`, состоящее из объектов `VPageBreak`, инкапсулирующих в себе информацию о вертикальных разрывах страницы

Таблица 3.18 *(окончание)*

Свойство	Описание
`Visible`	Управляет видимостью листа. Допустимые значения: `True` (рабочий лист видим на экране), `False` (не видим, т. е. скрыт, но его можно отобразить на экране командой **Формат | Лист | Отобразить**), `xlVeryHidden` (скрыт, и его можно отобразить на экране только программно)

Очистка диапазона с данными

Диапазон с данными возвращается свойством `UsedRange` рабочего листа. Поэтому комбинируя это свойство с методом `Clear` возможно очистить данные из рабочего листа.

```
Worksheets(1).UsedRange.Clear
```

Запрет на выделение пользователем диапазонов

Свойство `EnableSelection` рабочего листа управляет тем, может ли пользователь производить выделения на защищенном рабочем листе. У этого свойства имеются три допустимые значения: `xlNoSelection` (выделение запрещено), `xlNoRestrictions` (выделение разрешено), `xlUnlockedCells` (выделять можно только незаблокированные ячейки). В следующем коде (листинг 3.54) процедура `ProtectionOn` запрещает выделение любых ячеек на первом рабочем листе и, кроме того, устанавливает на нем защиту. Процедура `ProtectionOff` эту защиту снимает.

Листинг 3.54. Запрет на выделение пользователем диапазонов

```
Sub ProtectionOn()
   With Worksheets(1)
      .EnableSelection = xlNoSelection
      .Protect Contents:=True, UserInterfaceOnly:=True
   End With
End Sub

Sub ProtectionOff()
   Worksheets(1).Unprotect
End Sub
```

Проверка, установлена ли защита на содержании рабочего листа

Свойство `ProtectContents` возвращает значение `True`, если содержание рабочего листа защищено. Следующий код (листинг 3.55) проверяет наличие защиты содержания и в случае присутствия таковой, предлагает ее снять.

Листинг 3.55. Проверка, установлена ли защита на содержании рабочего листа

```
Sub DemoProtectContents()
   On Error Resume Next
   If Worksheets(1).ProtectContents Then
      MsgBox "Содержание рабочего листа защищено"
      Worksheets(1).Unprotect
   Else
      MsgBox "Содержание рабочего листа не защищено"
   End If
End Sub
```

Методы объекта *Worksheet*

Объект `Worksheet` обладает большой коллекцией методов, позволяющих производить широкий спектр действий над данными, начиная от выбора, перемещения и удаления, заканчивая проверкой правописания. В табл. 3.19 перечислим основные методы объекта `Worksheet`.

Таблица 3.19. *Основные методы объекта* `Worksheet`

Метод	Описание
`Activate`	Активизирует указанный рабочий лист
`Calculate`	Перевычисляет данные на рабочем листе
`ChartObjects`	Возвращает семейство `ChartObjects` всех вложенных в рабочий лист диаграмм
`CheckSpelling`	Проверка правописания в ячейках, примечаниях, заголовке и объектах, расположенных на рабочем листе
`CircleInvalid`	Обрамление некорректно введенных данных
`ClearArrows`	Удаление стрелок, демонстрирующих зависимости на рабочем листе
`ClearCircles`	Удаление обрамления с некорректно введенных данных
`Copy`	Копирует лист в другое местоположение той же книги

Таблица 3.19 *(окончание)*

Метод	Описание
`Delete`	Удаляет лист
`Move`	Перемещает лист
`OLEObjects`	Возвращает семейство `OLEObjects` всех внедренных OLE-объектов
`Paste`	Вставляет из буфера обмена на рабочий лист
`PasteSpecial`	Вставляет данные на рабочий лист, используя специальную вставку
`PivotTableWizard`	Создает сводную таблицу
`PrintOut`	Печатает содержимое рабочего листа
`PrintPreview`	Предварительный просмотр содержимого рабочего листа перед печатью
`Protect`	Устанавливает защиту на рабочий лист
`ResetAllPageBreaks`	Переустанавливает на рабочем листе все разрывы страницы
`SaveAs`	Сохраняет измененную страницу в отдельный файл
`Scenarios`	Возвращает либо объект `Scenario`, представляющий единственный сценарий, либо семейство `Scenarios` всех сценариев
`Select`	Выбирает указанный рабочий лист
`SetBackgroundPicture`	Устанавливает фоновый рисунок для рабочего листа
`ShowAllData`	Отображает все скрытые при помощи примененных фильтров данные
`Unprotect`	Снимает защиту с рабочего листа

Удаление рабочего листа без предупреждения пользователя

Для удаления рабочего листа без предупреждения пользователя надо временно при помощи свойства `DisplayAlerts` отключить отображение предупреждающего окна, затем удалить требуемый лист. После этого режим отображения предупреждений разумно снова включить.

```
Application.DisplayAlerts = False
Worksheets("Лист1").Delete
Application.DisplayAlerts = True
```

Смена фоновой картинки при щелчке правой кнопкой на ячейке

Метод `SetBackgroundPicture` задает фоновый рисунок рабочего листа.

```
SetBackgroundPicture(FileName)
```

Здесь *FileName* — параметр, указывающий имя файла с изображением. Причем, если в качестве значения этого параметра использовать пустую строку, то это равносильно удаления фонового рисунка. Тот факт, что фоновый рисунок допустимо изменять во время выполнения проекта, позволяет получать некоторые забавные спецэффекты. Например, следующий код (листинг 3.56) обеспечивает, что при щелчке правой кнопкой на ячейке **A1** фоновым рисунком становится изображение из файла Me.jpg, при щелчке правой кнопкой на ячейке **B1** — из файла Clouds.bmp, а при щелчке правой кнопкой на ячейке **C1** — фоновый рисунок удаляется (рис. 3.11). При этом файлы Me.jpg и Clouds.bmp должны быть расположены в том же каталоге, что и рабочая книга.

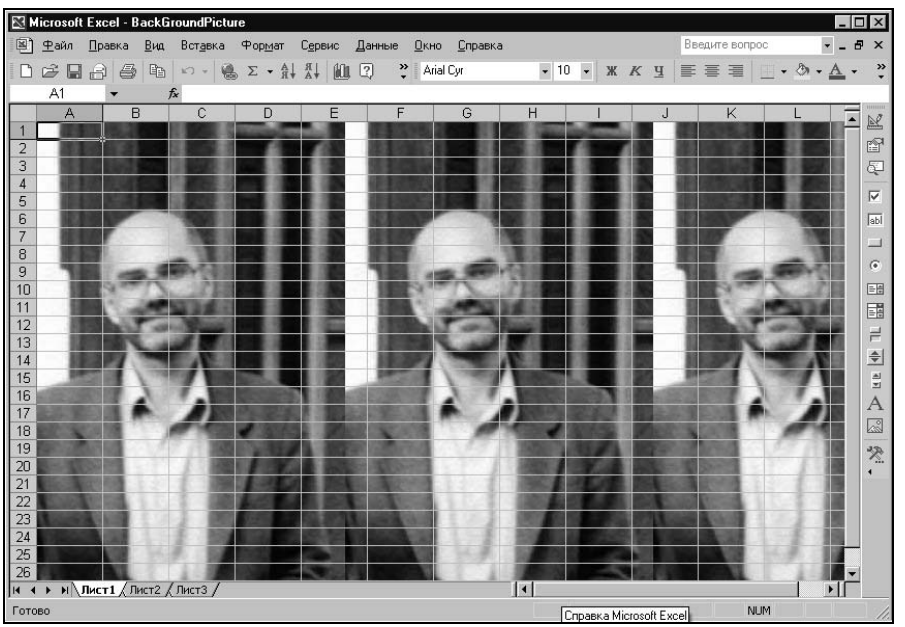


Рис. 3.11. Смена фоновой картинки

Листинг 3.56. Смена фоновой картинки при щелчке правой кнопкой на ячейке. Модуль рабочего листа

```
Private Sub Worksheet_BeforeRightClick(ByVal Target As Range, _
      Cancel As Boolean)
```

```
   If Target.Address = "$A$1" Then
      Cancel = True
      Worksheets(1).SetBackgroundPicture ThisWorkbook.Path & "\me.jpg"
   ElseIf Target.Address = "$B$1" Then
      Cancel = True
      Worksheets(1).SetBackgroundPicture ThisWorkbook.Path _
                                         & "\clouds.bmp"
   ElseIf Target.Address = "$C$1" Then
      Cancel = True
      Worksheets(1).SetBackgroundPicture ""
   End If
End Sub
```

Установка и снятие защиты с рабочего листа

Защита на рабочий лист устанавливается методом `Protect`, у которого имеется обширная коллекция необязательных параметров.

```
Protect(Password, DrawingObjects, Contents, Scenarios, UserInterfaceOnly,
↪AllowFormattingCells, AllowFormattingColumns, AllowFormattingRows,
↪AllowInsertingColumns, AllowInsertingRows, AllowInsertingHyperlinks,
↪AllowDeletingColumns, AllowDeletingRows, AllowSorting, AllowFiltering,
↪AllowUsingPivotTables)
```

- *Password* — задает пароль защиты;
- *DrawingObjects* — устанавливает защиту на графические объекты;
- *Contents* — определяет защиту на содержание ячеек;
- *Scenarios* — задает защиту на сценарии;
- *UserInterfaceOnly* — защита пользовательского интерфейса, но не макросов. Если этот параметр опущен, то защита применяется как к интерфейсу, так и макросам;
- *AllowFormattingCells* — разрешение форматировать ячейки;
- *AllowFormattingColumns* — разрешение форматировать столбцы;
- *AllowFormattingRows* — разрешение форматировать строки;
- *AllowInsertingColumns* — разрешение вставлять столбцы;
- *AllowInsertingRows* — разрешение вставлять строки;
- *AllowInsertingHyperlinks* — разрешение вставлять гиперссылки;
- *AllowDeletingColumns* — разрешение удалять столбцы;
- *AllowDeletingRows* — разрешение удалять строки;
- *AllowSorting* — разрешение сортировать данные;

- *AllowFiltering* — разрешение фильтровать данные;
- *AllowUsingPivotTables* — разрешение конструировать сводные таблицы.

Снятие защиты осуществляется методом

```
Unprotect(Password)
```

Здесь *Password* — пароль защиты.

Объект *Protection* (как определить, какая защита на рабочем листе установлена)

Объект Protection инкапсулирует в себе информацию о том, какая защита включена на рабочем листе. Этот объект возвращается свойством Protection объекта Worksheet. Его свойства, перечисленные в табл. 3.20, как раз и идентифицируют установленную защиту.

Таблица 3.20. *Свойства объекта* Protection

Свойство	Описание
AllowDeletingColumns	Разрешено удаление столбцов
AllowDeletingRows	Разрешено удаление строк
AllowFiltering	Разрешена фильтрация
AllowFormattingCells	Разрешено форматирование ячеек
AllowFormattingColumns	Разрешено форматирование столбцов
AllowFormattingRows	Разрешено форматирование строк
AllowInsertingColumns	Разрешена вставка столбцов
AllowInsertingHyperlinks	Разрешена вставка гиперссылок
AllowInsertingRows	Разрешена вставка строк
AllowSorting	Разрешена сортировка
AllowUsingPivotTables	Разрешено использование сводных таблиц

Кроме объекта Protection, имеется еще ряд свойств рабочего листа, определяющих тип защиты, не покрытых свойствами объекта Protection (табл. 3.21).

Таблица 3.21. Свойства, проверяющие установленную защиту

Свойство	Описание
ProtectContents	Свойство "только для чтения", проверяющее, установлена ли защита на содержимое рабочего листа
ProtectDrawingObjects	Свойство "только для чтения", проверяющее, установлена ли защита на графические объекты
ProtectScenarios	Свойство "только для чтения", проверяющее, установлена ли защита на сценарии
ProtectionMode	Свойство "только для чтения", проверяющее, установлена ли защита только пользовательского интерфейса

События объекта *Worksheet*

Объект Worksheet поддерживает множество событий, позволяющих отслеживать всевозможные действия пользователя от активизации листа до перехода по гиперссылке. В табл. 3.22 перечислены события, связанные с объектом Worksheet.

Таблица 3.22. События объекта Worksheet

Событие	Описание
Activate	Генерируется при активизации листа
BeforeDoubleClick	Генерируется перед двойным щелчком до того, как будет выполнено действие, ассоциированное по умолчанию с соответствующим объектом
BeforeRightClick	Генерируется перед щелчком правой кнопкой мыши до того, как будет выполнено действие, ассоциированное по умолчанию с соответствующим объектом
Calculate	Генерируется при пересчете данных
Change	Генерируется при изменениях данных
Deactivate	Генерируется при деактивизации листа
FollowHyperlink	Генерируется при переходе по гиперссылке
PivotTableUpdate	Генерируется при обновлении сводной таблицы
SelectionChange	Генерируется при смене выделения

Блокировка действий, связанных с событием по умолчанию

С теми или иными действиями пользователя связаны по умолчанию некоторые системные действия. Например, при нажатии правой кнопки мыши

на рабочем листе отображается стандартное контекстное меню. Естественно, возникает вопрос, как блокировать эти действия, если они не совпадают с бизнес-логикой вашего проекта. Да очень просто. В процедурах обработки событий часто присутствует параметр `Cancel`, который, если принимает значение `True`, как раз и блокирует эти действия по умолчанию, предлагаемые системой пользователю. Например, в следующем коде (листинг 3.57) блокируется отображение контекстного меню, появляющегося при нажатии правой кнопки мыши. Вместо него в месте на рабочем листе, где был произведен щелчок, добавляется надпись с порядковым номером этого щелчка.

Листинг 3.57. Блокировка отображения контекстного меню, появляющегося при нажатии правой кнопки мыши

```
Private Sub Worksheet_BeforeRightClick(ByVal Target As Range, _
                                       Cancel As Boolean)
   Cancel = True

   Static i As Integer
   Dim x As Integer
   Dim y As Integer
   x = Target.Left
   y = Target.Top
   ActiveSheet.Shapes.AddTextbox(msoTextOrientationHorizontal, _
               x, y, 30, 20).TextFrame.Characters.Text = i
   i = i + 1
End Sub
```

Автоматическое включение гиперссылки, содержащей ключевое слово, в папку *Избранное*

Событие `FollowHyperlink` можно использовать для автоматизации решения некоторых задач, возникающих при переходе по гиперссылке. Например, обеспечить автоматическое включение всех гиперссылок, содержащих ключевое слово, скажем `Microsoft`, в папку **Избранное**. Для этого достаточно воспользоваться функцией `InStr`, которая проверяет, является ли одна строка подстрокой другой строки, и в случае положительного результата проверки остается только методом `AddToFavorites` поместить гиперссылку в папку **Избранное** (листинг 3.58).

Листинг 3.58. Автоматическое включение гиперссылки, содержащей ключевое слово, в папку Избранное

```
Private Sub Worksheet_FollowHyperlink(ByVal Target As Hyperlink)
   If InStr(1, Target.Address, "Microsoft", 1) <> 0 Then
```

```
      Target.AddToFavorites
   End If
End Sub
```

Автоматическое переоформление таблицы при изменении в ней значений

Событие `Change` объекта `Worksheet` генерируется при изменении значений в диапазоне рабочего листа. Это событие позволяет автоматизировать переоформление таблицы при изменении в ней значений. Рассмотрим следующую бизнес-ситуацию. В диапазоне **B2:B11** расположены данные по объему продаж нафталина фирмой "Родные просторы". Необходимо выделить полужирным красным шрифтом те данные, которые соответствуют максимальному объему продаж, синим же цветом — минимальному. Все остальные данные выводятся черным шрифтом. Кроме того, ячейки, объем продаж в которых превышает средний объем, надо залить желтым цветом. Причем требуется обеспечить автоматическое переформатирование таблицы при изменении значений в ее ячейках (рис. 3.12). Для решения этой задачи как раз и пригодится событие `Change` объекта `Worksheet`, а как это делается, показано в следующем коде (листинг 3.59).

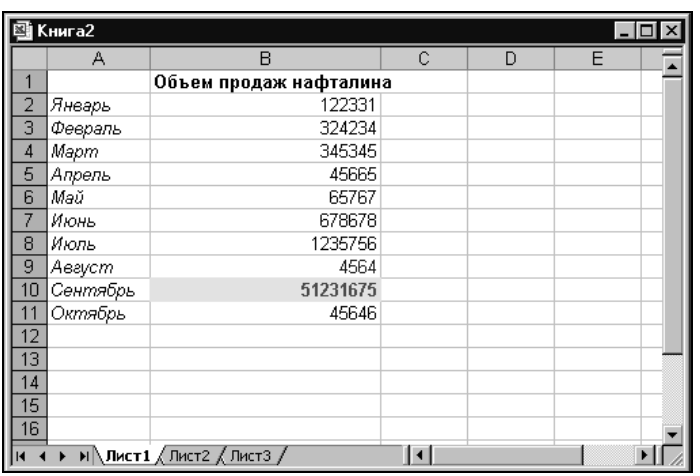


Рис. 3.12. Автоматическое переоформление таблицы при изменении в ней значений

Листинг 3.59. Автоматическое переоформление таблицы при изменении в ней значений

```
Private Sub Worksheet_Change(ByVal Target As Range)
   Dim rng As Range
   Dim c As Range
```

```
   Dim max As Double
   Dim min As Double
   Dim avr As Double
   Set rng = Range("B2:B11")
   If Not (Application.Intersect(Target, rng) Is Nothing) Then
      If Application.WorksheetFunction.CountA(rng) > 0 Then
         max = Application.WorksheetFunction.max(rng)
         min = Application.WorksheetFunction.min(rng)
         avr = Application.WorksheetFunction.Average(rng)
         For Each c In rng
            If c.Value = max Then
               c.Font.Bold = True
               c.Font.Color = RGB(255, 0, 0)
            ElseIf c.Value = min Then
               c.Font.Bold = False
               c.Font.Color = RGB(0, 0, 255)
            Else
               c.Font.Bold = False
               c.Font.Color = RGB(0, 0, 0)
            End If
            If c.Value > avr Then
               c.Interior.Color = RGB(255, 255, 0)
            Else
               c.Interior.ColorIndex = xlNone
            End If
         Next
      Else
        rng.Interior.ColorIndex = xlNone
      End If
   End If
End Sub
```

Объекты *Range* и *Selection*

В иерархии MS Excel объект `Range` *(диапазон)* идет сразу после объекта `Worksheet`. Объект `Range` является одним из ключевых объектов VBA. Объект `Selection` возникает в VBA двояко — либо как результат работы метода `Select`, либо при вызове свойства `Selection`. Тип получаемого объекта зависит от типа выделенного объекта. Чаще всего объект `Selection` принадлежит классу `Range`, и при работе с ним можно использовать свойства и методы объекта `Range`. Объект `Range` возвращается либо как элемент семейств `Range` или `Cells`, либо свойствами `Range`, `Cells` и `Offset`, либо методами `ActiveCell`, `Intersect` и `Union`.

Адресация ячеек

При работе с объектом `Range` необходимо помнить, как в MS Excel ссылаются на ячейку рабочего листа. Имеются два способа ссылки на ячейки рабочего листа:

- относительная адресация (табл. 3.23), т. е. когда начало координат, задающее нумерацию строк и столбцов, связывается с объектом, вызвавшим `Range`;
- абсолютная адресация (табл. 3.24).

Таблица 3.23. *Относительная адресация*

Формат	Описание
A1	Имя ячейки состоит из имени столбца (их 256 — A, B, ..., Z, AB, ..., HZ, IA, ..., IV) и номера (1, ..., 16 384). Например, **A1** или **C2**
R1C1	Адресация задается индексом строки и индексом столбца. Например, **R1C1** или **R2C3**

Таблица 3.24. *Абсолютная адресация*

Формат	Описание
A1	Признаком абсолютной адресации является знак "$", предшествующий имени строки (абсолютная адресация на строку) или столбца (абсолютная адресация на столбец). Например, `$A10`, `A$10` и `$A$10` задают абсолютную адресацию на столбец **A**, строку **10** и ячейку **A10** соответственно
R1C1	Указывается смещение по отношению к активной ячейке. Смещение приводится в квадратных скобках, причем знак указывает на направление смещения. Например, если активной ячейкой является **R2C3**, то `R[1]C[-1]` дает ссылку на ячейку **R3C2**

Адресация ячейки рабочего листа является лишь частью полного адреса ячейки, который в общем случае включает имя рабочего листа и адрес книги. При задании полного адреса за именем листа следует знак "!", а адрес книги заключается в скобки. Например:

```
A1
Лист2!A1
[МояКнига.XLS]Лист2!A1
```

В первой строке этого примера дана относительная ссылка на ячейку **A1** активного рабочего листа, во второй — на ячейку **A1** рабочего листа **Лист2** активной книги, а в третьей — на ячейку **A1** рабочего листа **Лист2** книги МояКнига.xls текущего рабочего каталога.

Задание групп строк и столбцов

Если в диапазоне указываются только имена столбцов или строк, то объект `Range` задает диапазон, состоящий из указанных столбцов или строк. Например, `Range("A:C")` задает диапазон, состоящий из столбцов **A**, **B** и **C**, а `Range("2:2")` — из второй строки. Другим способом работы со строками и столбцами являются свойства рабочего листа `Rows` и `Columns`, возвращающие семейства строк и столбцов. Например, столбцом **A** является `Columns(1)`, а второй строкой `Rows(2)`.

Связь объекта *Range* и свойства *Cells* объекта *Worksheet*

Ячейка — это частный случай диапазона, который состоит из единственной ячейки. Поэтому естественно, что объект `Range` позволяет работать как с диапазоном ячеек, так и с одной ячейкой.

Альтернативным способом работы с ячейкой является свойство `Cells` объекта `Worksheet`. Например, ячейку **A2** как объект можно описать двумя равносильными способами: `Range("A2")` и `Cells(1, 2)`.

В свою очередь ячейка, возвращаемая свойством `Cells`, используемым как параметр объекта `Range`, позволяет записывать диапазон в альтернативном виде, который иногда удобен для работы. В качестве примера этой формы записи диапазона приведем следующие две инструкции, возвращающие один и тот же диапазон:

```
Range("A2:C3")
Range(Cells(1,2), Cells(3,3))
```

Примечание

Диапазон так же, как и рабочий лист, обладает свойством `Cells`, которое, если используется без параметров, возвращает множество всех ячеек, входящих в диапазон. Если же оно используется с параметрами, то возвращает конкретную ячейку из диапазона. Например, в следующем примере значение `2` вводится в ячейку **C3**:

```
Range("B2:D4").Select
Selection.Cells(2, 2).Value = 2
```

Свойства объекта *Range*

Объект `Range` позволяет сочетать гибкость VBA и мощь рабочего листа. Огромное число встроенных функций рабочего листа существенно упрощают и делают более наглядным программирование на VBA. Свойства объекта `Range` позволяют управлять им от внешнего вида до автоматизации вычислений. В последующих разделах на содержательных примерах досконально разобраны следующие свойства объекта `Range`:

`Address`	`Formula`	`Offset`
`AllowEdit`	`FormulaArray`	`Orientation`
`Areas`	`FormulaHidden`	`Resize`
`Borders`	`FormulaLocal`	`Row`
`Cells`	`FormulaR1C1`	`RowHeight`
`Characters`	`FormulaR1C1Local`	`Rows`
`Column`	`HasFormula`	`ShrinkToFit`
`Columns`	`Height`	`Top`
`ColumnWidth`	`Hidden`	`UseStandardHeight`
`Comment`	`HorizontalAlignment`	`UseStandardWidth`
`Count`	`Hyperlinks`	`Value`
`CurrentRegion`	`Interior`	`VerticalAlignment`
`End`	`Left`	`Width`
`EntireColumn`	`Locked`	`Worksheet`
`EntireRow`	`Name`	`WrapText`
`Font`	`NumberFormat`	

Ввод или считывание значения из диапазона

Свойство `Value` объекта `Range` возвращает или устанавливает значение в ячейках диапазона. В первой инструкции данного примера переменной `x` присваивается значение из ячейки **C1**, во второй — в ячейку **C3** вводится строка `Отчет`, а в третьей — в каждую из ячеек диапазона **A1:B2** вводится `1`.

```
x = Range("C1").Value
Range("C3").Value = "Отчет"
Range("A1:B2").Value = 1
```

Ввод в диапазон массива значений

Диапазон можно заполнять не только поячеечно, но и за одну операцию, присваивая свойству `Value` диапазона либо переменную типа `Variant`, как

показано в первом примере (листинг 3.60), либо непосредственно массив значений, как продемонстрировано во втором примере (листинг 3.61).

Листинг 3.60. Ввод в диапазон массива значений. Первый пример

```
Sub DemoInput1()
   Dim s As Variant
   s = Array("1", "2")
   Range("A1:B1").Value = s
End Sub
```

Листинг 3.61. Ввод в диапазон массива значений. Второй пример

```
Sub DemoInput2()
   Dim t(8, 8) As Integer
   Dim i As Integer
   Dim j As Integer
   For i = 1 To 9
      For j = 1 To 9
         t(i — 1, j — 1) = i * j
      Next
   Next
   Range(Cells(1, 1), Cells(9, 9)).Value = t
End Sub
```

Поиск по шаблону подобных значений в диапазоне

Последовательный перебор ячеек диапазона и сравнение с помощью оператора `Like` возвращаемых значений свойства `Value` с шаблоном позволяет реализовывать поиск подобных значений в диапазоне. Например, в следующем коде (листинг 3.62) последовательно просматриваются все ячейки диапазона **A1:A100**. В тех из них, в которые входит значение `MS`, содержимое ячейки заменяется словом `Microsoft`, сама же ячейка заливается желтым цветом, в то время как все остальные ячейки — белым цветом.

Листинг 3.62. Поиск в диапазоне подобных значений

```
Dim c As Range
For Each c In [A1:A100]
   If c.Value Like "*MS*" Then
      c.Value = "Microsoft"
      c.Interior.Color = RGB(255, 255, 0)
```

```
   Else
      c.Interior.Color = RGB(255, 255, 255)
   End If
Next
```

Ввод или считывание формулы в ячейку в формате *A1*

Свойство `Formula` объекта `Range` возвращает или устанавливает формулу в диапазон в формате **A1**. Например, в следующем примере первая инструкция вводит в ячейку **C1** формулу `=$A$1+$B$1`, а вторая — в ячейку **C2** формулу `=SIN(A2)^2`.

```
Range("C1").Formula = "=$A$1+$B$1"
Range("C2").Formula = "=SIN(A2)^2"
```

Ввод или считывание формулы в ячейку в формате *R1C1*

Свойство `FormulaR1C1` объекта `Range` возвращает формулу в формате **R1C1**. Например, следующая инструкция вводит в ячейку **B1** формулу `=2*R3C2` в формате **R1C1**, или, что эквивалентно, формулу `=2*$B$3` в формате **A1**.

```
Range("B1").FormulaR1C1 = "=2*R3C2"
```

Ввод или считывание формулы локальной версии в ячейку в формате *A1*

Свойство `FormulaLocal` объекта `Range` возвращает формулу локальной версии в формате **A1**. Например, следующая инструкция вводит в ячейку **B2** формулу `=СУММ(C1:C4)`.

```
Range("B2"). FormulaLocal = "=СУММ(C1:C4)"
```

Ввод или считывание формулы локальной версии в ячейку в формате *R1C1*

Свойство `FormulaR1C1Local` объекта `Range` возвращает формулу локальной версии в формате **R1C1**. Например, следующая инструкция вводит в ячейку **B2** формулу `=СУММ(C1:C4)` в формате **R1C1**.

```
Range("B2").FormulaR1C1Local = "=СУММ(R1C3:R4C3)"
```

Ввод формулы массива в диапазон

Свойство `FormulaArray` объекта `Range` возвращает формулу диапазона в формате **A1**. В отличие от обыкновенной формулы рабочего листа, формула массива вводится на рабочем листе не нажатием клавиши <Enter>, а ком-

бинацией клавиш <Ctrl>+<Shift>+<Enter>. Например, следующая инструкция вводит в диапазон **E1:E3** формулу {=A1:A3*3}.

```
Range("E1:E3").FormulaArray = "=A1:A3*3"
```

Ввод формулы массива локальной версии в диапазон

При вводе формулы массива с функциями рабочего листа локальной версии формулу надо представить в формате **R1C1**, и вместо формулы локальной версии следует использовать формулу базовой версии. Например, следующая инструкция вводит в ячейку **D1** формулу {=СУММ($A$1:$B$1*3)}.

```
Range("D1").FormulaArray = "=SUM(R1C1:R1C2*3)"
```

Ввод формулы массива в диапазон с относительными ссылками на ячейки

Для ввода формулы с относительными ссылками на ячейки необходимо использовать относительную адресацию в формате **R1C1**. Например, ввод формулы {=СУММ(A1:B1*3)} в ячейку **D1** производится следующей инструкцией.

```
Range("D1").FormulaArray = "=SUM(RC[-3]:RC[-1]*3)"
```

Как узнать, спрятана ли формула на защищенном листе

Свойство `FormulaHidden` объекта `Range` возвращает значение `True`, если формула спрятана на защищенном листе.

Как узнать, имеется ли в ячейке формула

Свойство `HasFormula` объекта `Range` возвращает значение `True`, если во всех ячейках диапазона имеется по формуле, значение `False`, если формулы нет ни в одной из них, и значение `Null` — в оставшихся случаях. Например, следующий код проверяет наличие формулы в ячейке **C1** и, если таковой нет, вводит в эту ячейку формулу =1.

```
If Not Range("C1").HasFormula Then Range("C1").Formula = "=1"
```

Определение адреса ячейки

Свойство `Address` объекта `Range` возвращает адрес диапазона.

```
Address(RowAbsolute, ColumnAbsolute, ReferenceStyle, External,
RelativeTo)
```

- *RowAbsolute* — необязательный параметр, принимающий логические значения. Если значение параметра равно `True` или параметр опущен, то возвращается абсолютная ссылка на строку;

- *ColumnAbsolute* — необязательный параметр, принимающий логические значения. Если его значение равно `True` или параметр опущен, то возвращается абсолютная ссылка на столбец;
- *ReferenceStyle* — необязательный параметр. Допустимы два значения — `xlA1` и `xlR1C1`, если `xlA1` или `xlR1C1` опущены, то возвращается ссылка в формате **A1**;
- *External* — необязательный параметр, принимающий логические значения и определяющий, является ли ссылка внешней;
- *RelativeTo* — необязательный параметр. В случае если значения параметров *rowAbsolute* и *columnAbsolute* равны `False`, а *referenceStyle* — `xlR1C1`, то данный параметр определяет начальную ячейку диапазона, относительно которой производится адресация.

Следующий код (листинг 3.63), обрабатывающий событие `SelectionChange` объекта `Worksheet`, демонстрирует возвращаемые свойством `Address` значения при различных установках его параметров. Например, если на рабочем листе будет выбрана ячейка **A1**, то на экране отобразится окно со следующим сообщением:

```
$A$1
$A1
R1C1
R[-1]C[-1]
```

Листинг 3.63. Свойство `Address`. Модуль рабочего листа

```
Private Sub Worksheet_SelectionChange(ByVal Target As Range)
   MsgBox Target.Address() & vbCr & _
          Target.Address(RowAbsolute:=False) & vbCr & _
          Target.Address(ReferenceStyle:=xlR1C1) & vbCr & _
          Target.Address(ReferenceStyle:=xlR1C1, _
                         RowAbsolute:=False, _
                         ColumnAbsolute:=False, _
                         RelativeTo:=Worksheets(1).Cells(2, 2))
End Sub
```

Может ли ячейка быть редактирована на рабочем листе

Свойство "только для чтения" `AllowEdit` объекта `Range` возвращает значение `True`, если допустимо редактирование значений в указанном диапазоне, даже если на листе установлена защита.

Определения числа областей, из которых состоит данный диапазон

Свойство `Areas` объекта `Range` возвращает семейство `Areas` диапазонов, из которых состоит данный диапазон. Элементами семейства `Areas` является объект `Range`. Основным свойством этого семейства является свойство `Count`, возвращающее число элементов семейства. Например, следующий код (листинг 3.64), обрабатывающий событие `SelectionChange` объекта `Worksheet`, выводит в статусную строку число выделенных областей (рис. 3.13).

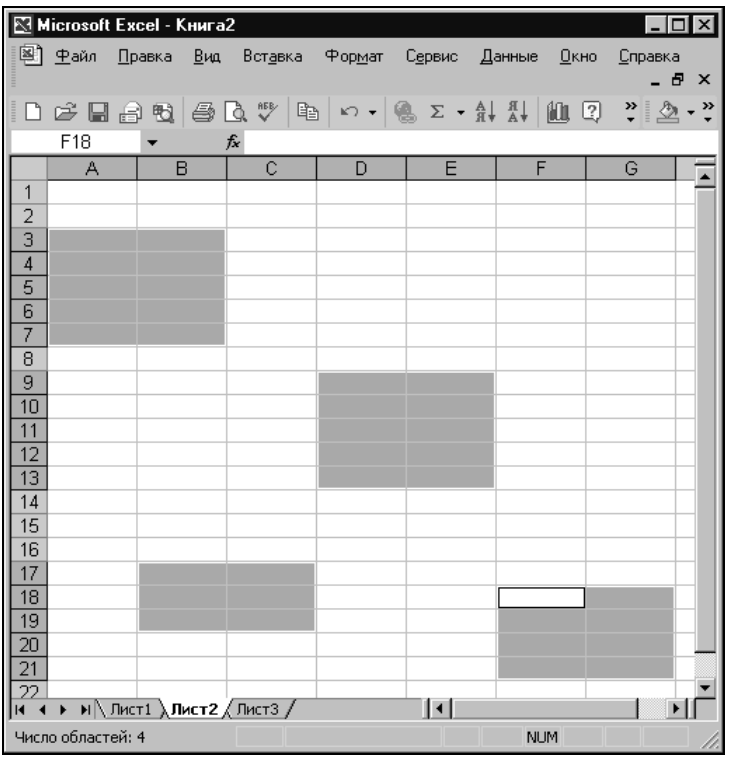


Рис. 3.13. Отображение в строке состояния числа выделенных областей

Листинг 3.64. Отображение в строке состояния числа выделенных областей. Модуль рабочего листа

```
Private Sub Worksheet_SelectionChange(ByVal Target As Range)
   Application.StatusBar = "Число областей: " & Target.Areas.Count
End Sub
```

Управление стилем границы диапазона и объекта *Border*

Свойство `Borders` объекта `Range` возвращает семейство `Borders`, элементы которого инкапсулируют данные об одной из граничных или диагональных линий данного диапазона. Допустимыми значениями индекса семейства `Borders` могут быть следующие константы `XlBordersIndex`: `xlDiagonalDown`, `xlDiagonalUp`, `xlEdgeBottom`, `xlEdgeLeft`, `xlEdgeRight`, `xlEdgeTop`, `xlInsideHorizontal` и `xlInsideVertical`. Каждая из этих границ представляет собой объект `Border`, свойства которого перечислены в табл. 3.25.

Таблица 3.25. *Свойства объекта* `Border`

Свойство	Описание
`Color`	Цвет границы, заданный при помощи RGB-модели
`ColorIndex`	Цвет границы, заданный индексом соответствующего элемента цветовой палитры
`LineStyle`	Задает стиль границы. Допустимыми значениями являются следующие константы `XlLineStyle`: `xlContinuous`, `xlDash`, `xlDashDot`, `xlDashDotDot`, `xlDot`, `xlDouble`, `xlSlantDashDot` и `xlLineStyleNone`
`Weight`	Устанавливает толщину границы. Допустимыми значениями являются следующие константы `XlBorderWeight`: `xlHairline`, `xlThin`, `xlMedium` и `xlThick`

Например, следующий код (листинг 3.65) задает верхнюю границу диапазона **A2:C2** в виде толстой линии красного цвета, а его нижнюю границу — в виде зеленой пунктирной линии средней толщины.

Листинг 3.65. Конструирование границы специфицированного диапазона

```
Sub DemoBorders()
   Dim rgn As Range
   Set rgn = Range("A2:C2")
   With rgn.Borders(xlEdgeTop)
      .LineStyle = xlContinuous
      .Weight = xlThick
      .Color = RGB(255, 0, 0)
   End With
   With rgn.Borders(xlEdgeBottom)
      .LineStyle = xlDash
```

```
      .Weight = xlMedium
      .Color = RGB(0, 255, 0)
   End With
End Sub
```

Если все компоненты границы имеют одни и те же параметры, то для установки их значения можно воспользоваться не элементами, а всем семейством `Borders`, как это, например, делается в следующей инструкции для создания границы синего цвета у выделенной области.

```
Selection.Borders.Color = RGB(0, 0, 255)
```

Функции *RGB* и *QBColor*

Коды цветов в VBA часто задаются числами в форме шестнадцатеричной системы счисления. Вместо прямого указания шестнадцатеричного кода цвета, довольно часто цвет удобнее задавать, используя функции `RGB` и `QBColor`. Функция `RGB` позволяет получить любой цвет смешением красного, зеленого и синего компонентов различной интенсивности.

```
RGB(Red, Green, Blue)
```

- *Red* — целое число из диапазона от 0 до 255, указывающее красный компонент цвета;
- *Green* — целое число из диапазона от 0 до 255, указывающее зеленый компонент цвета;
- *Blue* — целое число из диапазона от 0 до 255, указывающее синий компонент цвета.

В табл. 3.26 приведены значения параметров функции `RGB` для получения стандартных цветов.

Таблица 3.26. *Значения параметров функции `RGB` для получения стандартных цветов*

Цвет	Red	Green	Blue
Черный	0	0	0
Синий	0	0	255
Зеленый	0	255	0
Голубой	0	255	255
Красный	255	0	0
Розовый	255	0	255

Таблица 3.26 *(окончание)*

Цвет	Red	Green	Blue
Желтый	255	255	0
Белый	255	255	255

Функция `QBColor` возвращает шестнадцать основных цветов в зависимости от значения параметра (табл. 3.27).

```
QBColor(color)
```

Здесь парметр *`color`* принимает целые числа из диапазона от 0 до 15.

Таблица 3.27. *Соответствие между цветами и значением параметра функции* `QBColor`

Число	Цвет	Число	Цвет
0	Черный	8	Серый
1	Синий	9	Светло-синий
2	Зеленый	10	Светло-зеленый
3	Голубой	11	Светло-голубой
4	Красный	12	Светло-красный
5	Розовый	13	Светло-розовый
6	Желтый	14	Светло-желтый
7	Белый	15	Насыщенный белый

Доступ к отдельным ячейкам диапазона

Свойство `Cells` объекта `Range`, использованное без индексов, возвращает все ячейки диапазона, а с индексами — конкретную ячейку, специфицированную либо ее номером (один параметр), либо местоположением (два параметра).

Например, в следующем коде (листинг 3.66) в диапазоне **B1:C3** все положительные значения заменяются на 1, а отрицательные — на −1.

Листинг 3.66. Все ячейки диапазона

```
Dim c As Range
For Each c In Range("B1:C3").Cells
```

```
  If c.Value > 0 Then
     c.Value = 1
  ElseIf c.Value < 0 Then
     c.Value = -1
  End If
Next
```

Ту же задачу можно решить, используя свойство `Cells` с двумя параметрами, возвращающее ячейку, стоящую на пересечении указанной строчки и столбца. При этом надо помнить, что свойство `Cells` задает относительное расположение ячеек по отношению к диапазону (листинг 3.67).

Листинг 3.67. Относительное местоположение ячеек

```
Dim i As Integer
Dim j As Integer
For i = 1 To Range("B1:C3").Rows.Count
   For j = 1 To Range("B1:C3").Columns.Count
      If Range("B1:C3").Cells(i, j).Value > 0 Then
         Range("B1:C3").Cells(i, j).Value = 1
      ElseIf Range("B1:C3").Cells(i, j).Value < 0 Then
         Range("B1:C3").Cells(i, j).Value = -1
      End If
   Next
Next
```

Если требуется задать абсолютное местоположение ячеек, то надо воспользоваться свойством `Cells` рабочего листа, например, как это делается для той же самой задачи в листинге 3.68.

Листинг 3.68. Абсолютное местоположение ячеек

```
Dim i As Integer
Dim j As Integer
For i = 1 To 3
   For j = 2 To 3
      If Cells(i, j).Value > 0 Then
         Cells(i, j).Value = 1
      ElseIf Cells(i, j).Value < 0 Then
         Cells(i, j).Value = -1
      End If
   Next
Next
```

Объект *Characters* (как форматировать часть содержимого ячейки)

Свойство `Characters` объекта `Range` возвращает объект `Characters`, представляющий собой строку указанной длины от указанного символа. Часто применяется, когда надо форматировать содержание не всей ячейки, а только ее часть.

```
Characters(Start, Length)
```

- *Start* — необязательный параметр, задающий номер первого возвращаемого символа из данной строки;
- *Length* — необязательный параметр, указывающий число возвращаемых символов.

В следующем примере (листинг 3.69) в ячейку **A1** выводится строка `Андрей Гарнаев`, причем первая часть этой строки `Андрей` выводится полужирным шрифтом зеленого цвета высотой 16, а вторая ее часть `Гарнаев` — курсивным шрифтом красного цвета высотой 14 (рис. 3.14).

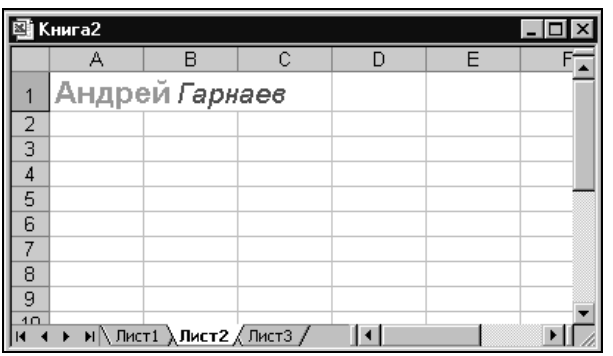


Рис. 3.14. Форматирование части содержимого ячейки

Листинг 3.69. Форматирование части содержимого ячейки

```
With Range("A1")
    .Value = "Андрей Гарнаев"
    .Characters(1, 6).Font.Bold = True
    .Characters(1, 6).Font.Size = 16
    .Characters(1, 6).Font.Color = RGB(0, 255, 0)
    .Characters(8, 7).Font.Italic = True
    .Characters(8, 7).Font.Size = 14
    .Characters(8, 7).Font.Color = RGB(255, 0, 0)
End With
```

Номер первого столбца и строки первой области диапазона

Свойства "только для чтения" `Column` и `Row` объекта `Range` возвращают номер первого столбца и строки первой области диапазона. В частности, их можно использовать для определения номера первой строки и столбца данного диапазона, например, как это делается в следующем коде (листинг 3.70).

Листинг 3.70. Определение номера первой строки и столбца выделенного диапазона

```
Dim c As Integer
Dim r As Integer
c = Selection.Column
r = Selection.Row
Msgbox r & vbCr & c
```

Семейство всех столбцов и строк данного диапазона

Свойства `Columns` и `Rows` объекта `Range` возвращают семейства `Columns` и `Rows`, представляющие собой наборы всех столбцов и строк данного диапазона. Основным свойством этих семейств является свойство `Count`, возвращающее число элементов семейства. В следующем примере в ячейки выделенного диапазона вставляется таблица умножения:

```
Dim i As Integer
Dim j As Integer
For i = 1 To Selection.Rows.Count
   For j = 1 To Selection.Columns.Count
      Selection.Cells(i, j).Value = i * j
   Next j
Next i
```

Нахождение номера последнего столбца и строки диапазона

Свойства `Column` и `Row` объекта `Range` могут быть использованы для нахождения номера последнего столбца и строки диапазона. Для этого диапазон надо разбить на отдельные столбцы и строки, используя свойства `Columns` и `Rows`, а затем, применяя уже свойства `Column` и `Row`, найти соответствующие номера для последних элементов семейств `Columns` и `Rows` (листинг 3.71).

Листинг 3.71. Определение номера последнего столбца и строки диапазона

```
Private Sub Worksheet_SelectionChange(ByVal Target As Range)
   Dim c As Integer
   Dim r As Integer
   c = Target.Columns(Target.Columns.Count).Column
   r = Target.Rows(Target.Rows.Count).Row
   MsgBox "Столбец: "& c & vbCr _
          & "Строка: " & r
End Sub
```

Установка ширины и высоты как самого диапазона, так и всех его строк и столбцов

Свойства `ColumnWidth` и `RowHeight` объекта `Range` возвращают и устанавливают ширину и высоту всех строк и столбцов диапазона. Свойства `Width` и `Height` объекта `Range` возвращают и устанавливают ширину и высоту всего диапазона. Например, следующие инструкции устанавливают ширину первого столбца и высоту первой строки в два раза больше, чем они были первоначально.

```
With ActiveSheet.Rows(1)
   .RowHeight = .RowHeight * 2
End With
With ActiveSheet.Columns(1)
   .ColumnWidth = .ColumnWidth * 2
End With
```

Число ячеек диапазона

Свойство `Count` объекта `Range` возвращает число ячеек, из которых состоит диапазон. Например, в следующем сообщение отобразится значение 9.

```
MsgBox Range("C1:E3").Count
```

Объект *Comment* (создание комментариев к диапазону)

Свойство `Comment` объекта `Range` возвращает объект `Comment` *(примечание)*, которое при отображении на экране связано с правым верхним углом диапазона. Вручную на рабочем листе примечание добавляется к диапазону выбором команды **Вставка | Примечание**.

В коде примечание можно создать методом `AddComment` объекта `Range`.

```
AddComment(Text)
```

Здесь *Text* — текст, вводимый в примечание. Объект `Comment` является элементом семейства `Comments`.

В табл. 3.28 перечислены свойства объекта Comment, а в табл. 3.29 — его методы.

Таблица 3.28. *Свойства объекта* Comment

Свойство	Описание
Author	Возвращает автора примечания
Shape	Возвращает объект Shape, задающий форму примечания
Parent	Возвращает ссылку на диапазон, для которого примечание было создано
Visible	Устанавливает, отображается ли примечание при активизации диапазона

Таблица 3.29. *Методы объекта* Comment

Свойство	Описание
Delete	Удаляет примечания
Next	Возвращает следующее примечание
Previous	Возвращает предыдущее примечание
Text	Возвращает текст, отображаемый в примечании

Итак, метод Text задает текст, выводимый в примечании.

```
Text(Text, Start, Overwrite)
```

- *Text* — необязательный параметр, который задает текст, выводимый в качестве примечания;
- *Start* — необязательный параметр, который определяет, с какого символа вводится текст в уже существующее примечание. Если параметр опущен, то из примечания удаляется весь ранее введенный текст;
- *Overwrite* — необязательный параметр, принимающий логические значения. Если он равен True, то вводимый текст записывается вместо уже существующего, а если False, то — добавляется к уже существующему.

В качестве примера рассмотрим следующую процедуру обработки события Open объекта Workbook, который при открытии рабочей книги в ячейке **G4** первого рабочего листа создает комментарий (листинг 3.72, *а*). При этом если в этой ячейке уже существовал комментарий, то он удаляется и затем создается по-новому. Процедура же обработки события BeforeDoubleClick

объекта Worksheet обеспечивает отображение или скрытие примечания при двойном щелчке на листе (листинг 3.72, *б*).

Листинг 3.72, *а*. Создание примечания при открытии книги. Модуль ЭтаКнига

```
Private Sub Workbook_Open()
   Set r = Worksheets(1).Range("G4").Comment
   If Not IsObject(r) Then
      Worksheets(1).Range("G4").Comment.Delete
   End If
   Worksheets(1).Range("G4").AddComment Text:="Очень важно!"
End Sub
```

Листинг 3.72, *б*. Отображение или скрытие примечания при двойном щелчке на листе. Модуль рабочего листа

```
Private Sub Worksheet_BeforeDoubleClick(ByVal Target As Range, _
                                        Cancel As Boolean)
   If Range("G4").Comment.Visible Then
      Range("G4").Comment.Visible = False
   Else
      Range("G4").Comment.Visible = True
   End If
End Sub
```

Определение текущего диапазона

Свойство CurrentRegion объекта Range возвращает *текущий диапазон* (current region). Текущий диапазон — это максимальный диапазон, который окружен любой комбинацией пустых строк и столбцов, ячейки которого не пусты, а сам он содержит в себе данный диапазон.

Например, предположим, в диапазоне **A1:E12** имеется таблица, которая по столбцу **F** и 13-й строке окружена пустыми ячейками. Тогда следующая инструкция вызовет выделение всей таблицы, т. е. диапазона **A1:E12**.

```
Cells(1, 1).CurrentRegion.Select
```

Усложним немножко задачу. Пусть у этой таблицы первая строка отведена под заголовок. Тогда, для того чтобы выделить всю таблицу без заголовка, достаточно применить следующие инструкции

```
Dim r As Range
Set r = Cells(1, 1).CurrentRegion
r.Offset(1, 0).Resize(r.Rows.Count – 1, r.Columns.Count).Select
```

В следующем примере переменной y присваивается значение, равное числу строк в текущем диапазоне, содержащем ячейку **A1**.

```
y = Range("A1").CurrentRegion.Rows.Count
```

Нахождение крайней ячейки диапазона в указанном направлении

Свойство End объекта Range возвращает крайнюю ячейку в указанном направлении в диапазоне, содержащем заданный диапазон.

```
End(Direction)
```

Здесь допустимыми значениями параметра *Direction* может быть одна из следующих констант XlDirection: xlToLeft, xlToRight, xlUp и xlDown.

Например, в следующем примере строка x будет введена в ячейку **A4**, а y — в ячейку **B2**:

```
Range("A1:B4").Value = "Тест"
Range("A1:B4").Select
Selection.End(xlDown).Value = "x"
Range("A2").End(xlToRight).Value = "y"
```

Нахождение строки и столбца, содержащих данную ячейку

Свойства EntireColumn и EntireRow объекта Range возвращают столбец и строку, содержащие данную ячейку. Следующий код, например, очищает всю строку, содержащую данную ячейку.

```
ActiveCell.EntireRow.Clear
```

Как спрятать строку или столбец

Свойство Hidden объекта Range возвращает значение True, если спрятаны строки или столбцы. Например, следующая процедура (листинг 3.73), обрабатывающая событие BeforeDoubleClick объекта Worksheet, обеспечивает последовательное спрятывание и отображение второго столбца при двойном щелчке.

Листинг 3.73. Последовательное спрятывание и отображение столбца при двойном щелчке. Модуль рабочего листа

```
Private Sub Worksheet_BeforeDoubleClick(ByVal Target As Range, _
                                        Cancel As Boolean)
  If Columns(2).Hidden Then
     Columns(2).Hidden = False
```

```
  Else
     Columns(2).Hidden = True
  End If
End Sub
```

Горизонтальное и вертикальное выравнивание содержимого диапазона

Свойства `HorizontalAlignment` и `VerticalAlignment` объекта `Range` обеспечивают горизонтальное и вертикальное выравнивание содержимого диапазона.

Допустимыми значениями свойства `HorizontalAlignment` являются:

- `xlGeneral` — обычное выравнивание, зависящее от типа вводимых значений;
- `xlCenter` — выравнивание по центру;
- `xlRight` — выравнивание по правому краю;
- `xlLeft` — выравнивание по левому краю;
- `xlJustify` — выравнивание по ширине;
- `xlCenterAcrossSelection` — выравнивание по центру в выделенном диапазоне;
- `xlFill` — выравнивание с заполнением.

Допустимыми значениями свойства `VerticalAlignment` являются:

- `xlBottom` — выравнивание по нижнему краю;
- `xlCenter` — выравнивание по центру;
- `xlJustify` — выравнивание по высоте;
- `xlTop` — выравнивание по верхнему краю.

Например, следующий код обеспечивает выравнивание по центру как по горизонтали, так и по вертикали содержимого ячеек **A1** и **B1**.

```
Range("A1") = "Июнь"
Range("B1") = "Июль"
Range("A1:B1").HorizontalAlignment = xlCenter
Range("A1:B1").VerticalAlignment = xlCenter
```

Объект *Hyperlink* (задание гиперссылки)

Свойство `Hyperlinks` объекта `Range` возвращает семейство `Hyperlinks`, состоящее из объектов `Hyperlink`, инкапсулирующих в себе информацию о гиперссылках.

У семейства `Hyperlinks` наиболее важными свойствами являются свойства `Count` и `Item`, возвращающие число элементов и конкретный элемент семей-

ства. Кроме того, в этом семействе имеются два метода: метод `Delete`, который удаляет гиперссылку, и метод `Add`, который создает новый элемент семейства.

`Add(`*`Anchor`*`, `*`Address`*`, `*`SubAddress`*`, `*`ScreenTip`*`, `*`TextToDisplay`*`)`

- *`Anchor`* — обязательный параметр, идентифицирующий местоположение гиперссылки;
- *`Address`* — обязательный параметр, задающий адрес гиперссылки;
- *`SubAddress`* — необязательный параметр, определяющий локальный адрес по отношению к адресу документа;
- *`ScreenTip`* — необязательный параметр, задающий текст всплывающей подсказки;
- *`TextToDisplay`* — необязательный параметр, задающий текст гиперссылки.

Основные свойства объекта `Hyperlink` перечислены в табл. 3.30, а методы — в табл. 3.31.

Таблица 3.30. *Основные свойства объекта* `Hyperlink`

Свойство	Описание
`Address`	Возвращает адрес документа, в который происходит переход
`EmailSubject`	Возвращает строку, отображаемую в разделе **Тема сообщения**, при отсылке электронной почты
`Name`	Возвращает имя объекта
`Parent`	Возвращает ссылку на объект (диапазон), в котором расположена гиперссылка
`Range`	Возвращает ссылку на диапазон, в котором расположена гиперссылка
`ScreenTip`	Возвращает текст всплывающей подсказки
`Shape`	Возвращает объект `Shape`, к которому присоединена гиперссылка
`SubAddress`	Возвращает локальный адрес по отношению к адресу документа
`TextToDisplay`	Возвращает текст гиперссылки

Таблица 3.31. *Методы объекта* `Hyperlink`

Метод	Описание
`AddToFavorites`	Добавляет гиперссылку в папку **Избранное**

Таблица 3.31 *(окончание)*

Метод	Описание
`CreateNewDocument`	Открывает новый документ, связанный с указанной гиперссылкой
`Delete`	Удаляет гиперссылку
`Follow`	Отображает указанный документ, если он уже загружен. В противном случае производит его загрузку и отображение. `Follow (`*`NewWindow`*`, `*`AddHistory`*`, `*`ExtraInfo`*`, `*`Method`*`, ↳`*`HeaderInfo`*`)` ♦ *`NewWindow`* — необязательный параметр, если он принимает значение `True`, то документ отображается в новом окне, если же его значение равно `False`, то в том же самом; ♦ *`AddHistory`* — не используется, зарезервирован для будущих версий MS Excel; ♦ *`ExtraInfo`* — необязательный параметр, содержащий дополнительную информацию, используемую для HTTP-протокола; ♦ *`Method`* — необязательный параметр, специфицирующий то, как передается информация из параметра *`ExtraInfo`*; ♦ *`HeaderInfo`* — необязательный параметр, содержащий заголовочную информацию, передаваемую при HTTP-запросе

Например, следующая инструкция в ячейке **A1** создает гиперссылку `MS` со всплывающей подсказкой `Microsoft` для открытия Web-страницы **http://www.microsoft.com**.

```
With Worksheets(1)
    .Hyperlinks.Add Anchor:=.Range("A1"), _
                    Address:="http://www.microsoft.com", _
                    ScreenTip:="Microsoft", _
                    TextToDisplay:="MS"
End With
```

Объект *Font* (задание шрифта)

Свойство `Font` объекта `Range` возвращает объект `Font`, представляющий собой шрифт. В табл. 3.32 приведены свойства объекта `Font`.

Таблица 3.32. *Свойства объекта* `Font`

Свойство	Описание
`Bold`	Определяет, является ли шрифт полужирным
`Color`	Задает цвет шрифта в соответствии с RGB-моделью

Таблица 3.32 *(окончание)*

Свойство	Описание
`ColorIndex`	Задает индексированный цвет в соответствии с текущей палитрой цветов
`FontStyle`	Задает стиль шрифта, заданный в словесной форме. Допустимы значения: `Regular` (обычный), `Bold` (полужирный), `Italic` (курсив), `Bold Italic` (полужирный курсив)
`Italic`	Определяет, является ли шрифт курсивным
`Name`	Строка, указывающая имя шрифта, например `"Arial Cyr"`
`Size`	Размер шрифта
`Strikethrough`	Устанавливает, имеется ли линия по центру, как будто текст перечеркнут
`Superscript`	Устанавливает, используется ли текст как верхний индекс
`Subscript`	Устанавливает, используется ли текст как нижний индекс
`Underline`	Задает тип подчеркивания. Допустимы значения: `xlNone` (нет подчеркивания), `xlSingle` (одинарное, по значению), `xlDouble` (двойное, по значению), `xlSingleAccounting` (одинарное, по ячейке), `xlDoubleAccounting` (двойное, по ячейке)

Например, в следующем примере устанавливается для диапазона **A1:B2** полужирный шрифт красного цвета с высотой символов `14`.

```
With Range("A1:B2").Font
    .Size = 14
    .Bold = True
    .Color = RGB(255, 0, 0)
End With
```

Объект *Interior* (заливка диапазона)

Свойство `Interior` объекта `Range` возвращает объект `Interior`, представляющий собой объект, инкапсулирующий данные о заливке диапазона. В табл. 3.33 приведены свойства объекта `Interior`.

Таблица 3.33. *Свойства объекта* `Interior`

Свойство	Описание
`Color`	Задает цвет заливки в соответствии с RGB-моделью
`ColorIndex`	Задает индексированный цвет заливки в соответствии с текущей палитрой цветов

Таблица 3.33 *(окончание)*

Свойство	Описание
`Pattern`	Задает узор заливки. Допустимыми являются следующие значения: `xlPatternAutomatic`, `xlPatternChecker`, `xlPatternCrissCross`, `xlPatternDown`, `xlPatternGray16`, `xlPatternGray25`, `xlPatternGray50`, `xlPatternGray75`, `xlPatternGray8`, `xlPatternGrid`, `xlPatternHorizontal`, `xlPatternLightDown`, `xlPatternLightHorizontal`, `xlPatternLightUp`, `xlPatternLightVertical`, `xlPatternNone`, `xlPatternSemiGray75`, `xlPatternSolid`, `xlPatternUp`, `xlPatternVertical`
`PatternColor`	Задает цвет узора в соответствии с RGB-моделью
`PatternColorIndex`	Задает индексированный цвет узора в соответствии с текущей палитрой цветов

Например, в следующем примере (листинг 3.74) устанавливается красная заливка диапазона **A1:D5** с синим клетчатым узором (рис. 3.15).

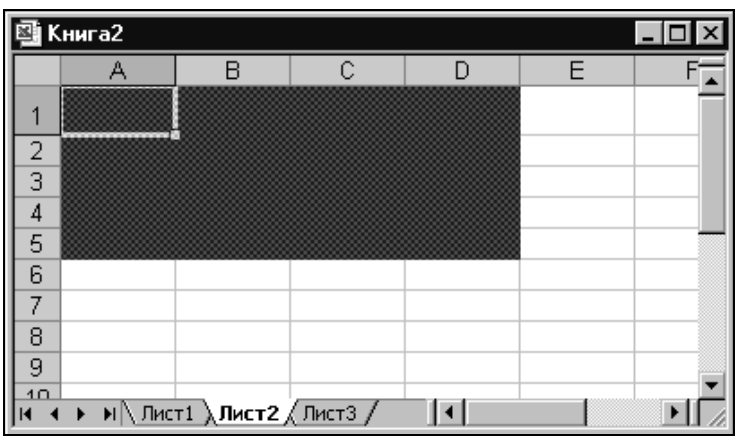


Рис. 3.15. Заливка диапазона

Листинг 3.74. Заливка диапазона

```
With Range("A1:D5")
   .Interior.Color = RGB(255, 0, 0)
   .Interior.Pattern = xlPatternChecker
   .Interior.PatternColor = RGB(0, 0, 255)
End With
```

Отмена заливки диапазона

Для отмены заливки диапазона надо установить значение свойства `ColorIndex` объекта `Interior` равным `xlNone`, например, как это делается в следующей инструкции для ячейки **A1**.

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

Местоположение диапазона и привязка расположения объекта к конкретной ячейке

Свойства `Top`, `Left` встроенного объекта и, в частности, объекта `Range`, возвращают координаты его левого верхнего угла. Эти свойства позволяют привязывать расположения объекта к конкретному диапазону или другому объекту. Например, следующий код (листинг 3.75) размещает диаграмму так, чтобы ее левый верхний угол и размеры совпадали с соответствующими атрибутами диапазона **C2:G5**.

Листинг 3.75. Привязка расположения объекта к конкретной ячейке

```
With Worksheets(1)
   .ChartObjects(1).Left = .Range("C2:G5").Left
   .ChartObjects(1).Top = .Range("C2:G5").Top
   .ChartObjects(1).Width = .Range("C2:G5").Width
   .ChartObjects(1).Height = .Range("C2:G5").Height
End With
```

Разрешение редактирования содержимого ячеек на защищенном рабочем листе

Свойство `Locked` объекта `Range`, установленное равным значению `False`, обеспечивает пользователю возможность редактирования содержимого ячеек даже на защищенном рабочем листе. Например, следующая процедура (листинг 3.76), обрабатывающая событие `Open` объекта `Workbook`, вводит в ячейку **C1** формулу `=A1+B1` и защищает по паролю `stop` все ячейки листа, кроме ячеек **A1** и **B1**, в которые пользователь может вводить любые значения.

Листинг 3.76. Разрешение редактирования содержимого ячеек на защищенном рабочем листе. Модуль `ЭтаКнига`

```
Private Sub Workbook_Open()
   On Error Resume Next
   Worksheets(1).Range("C1").Formula = "=A1+B1"
```

```
    Worksheets(1).Range("A1:B1").Locked = False
    Worksheets(1).Protect "stop"
End Sub
```

Имя диапазона

Свойство `Name` объекта `Range` возвращает или устанавливает имя диапазона. Вручную на рабочем листе имя диапазону присваивается выбором команды **Вставка | Имя | Присвоить**. В следующем примере ячейке **A1** присваивается имя `x`, ячейке **B1** — имя `y`, а в ячейку **C1** вводится формула `=x+y`.

```
Range("A1").Name = "x"
Range("B1").Name = "y"
Range("C1").Formula = "=x+y"
```

В следующем примере диапазону **A1:B2** присваивается имя `Итоги`, а в ячейку **C1** вводится формула `=СУММ(Итоги)`.

```
Range("A1:B2").Name = "Итоги"
Range("C1").FormulaLocal = "=СУММ(Итоги)"
```

Установка числового формата

Свойство `NumberFormat` объекта `Range` устанавливает числовой формат. Например, следующие инструкции устанавливают (листинг 3.77):

- в ячейке **A1** — общий формат,
- в ячейке **A2** — числовой формат, отображающий три знака после десятичной точки, например `12.000`;
- в ячейке **A3** — формат времени с двоеточием в качестве разделителя и по два знака, отведенных под часы, минуты и секунды, например `02:12:55`;
- в ячейке **A4** — формат даты, причем два знака отводится под день, три буквы — под месяц и четыре цифры — под год, например `01 июл 2002`.

Листинг 3.77. Установка числового формата

```
Range("A1").NumberFormat = "General"
Range("A2").NumberFormat = "0.000"
Range("A3").NumberFormat = "hh:mm:ss"
Range("A4").NumberFormat = "d mmm yyyy"
```

Нахождение диапазона, сдвинутого относительно данного на указанное число строк и столбцов

Свойство `Offset` объекта `Range` возвращает объект `Range`, который сдвинут по отношению к данному на указанное число строк и столбцов.

```
Offset(RowOffset, ColumnOffset)
```

- *`RowOffset`* — необязательный параметр, указывающий сдвиг по строкам;
- *`ColumnOffset`* — необязательный параметр, указывающий сдвиг по столбцам.

Например, в следующем коде в активную ячейку, например **A1**, вводится значение `3`, в ячейку, сдвинутую от нее на один столбец (в данном случае **B1**), вводится значение `4`, а в ячейку, сдвинутую еще на один столбец (в данном случае **C1**), вводится формула, находящая сумму содержимого двух первых ячеек.

```
ActiveCell.Value = 3
ActiveCell.Offset(0, 1).Value = 4
ActiveCell.Offset(0, 2).FormulaR1C1 = "=R[0]C[-2]+R[0]C[-1]"
```

Задание угла, под которым выводится текст в диапазоне

Свойство `Orientation` объекта `Range` устанавливает угол, под которым выводится текст в диапазоне. Допустимыми значениями являются либо угол поворота текста в градусах от −90 до 90°, либо одна из следующих постоянных:

- `xlDownward` — выравнивание по левому краю сверху вниз, соответствует углу −90°;
- `xlHorizontal` — выравнивание по горизонтали, соответствует нулевому углу;
- `xlUpward` — выравнивание по правому краю снизу вверх, соответствует углу 90°;
- `xlVertical` — выравнивание по вертикали, нет соответствия в градусах.

Например, в следующем коде (листинг 3.78) в ячейке **A1** текст выводится под углом 45°, а в ячейке **B1** — под углом −45° (рис. 3.16).

Листинг 3.78. Задание угла вывода текста

```
Range("A1:B1").Font.Size = 16
Range("A1").Orientation = 45
Range("A1").Value = "Вверх"
Range("B1").Orientation = -45
Range("B1").Value = "Вниз"
```



Рис. 3.16. Задание угла вывода текста

Переопределение размеров диапазона

Свойство `Resize` объекта `Range` возвращает диапазон с измененными размерами.

```
Resize(RowSize, ColumnSize)
```

- *RowSize* — необязательный параметр, указывающий новое число строк;
- *ColumnSize* — необязательный параметр, указывающий новое число столбцов.

Например, предположим, что на рабочем листе имеется таблица с левым верхним углом, расположенным в ячейке **A1**, причем первая строка таблицы является заголовком. Тогда следующие инструкции создают красную заливку для заголовка таблицы и желтую — для ее тела.

```
Dim tbl As Range
Set tbl = Range("A1").CurrentRegion
Set tbl = tbl.Resize(1, tbl.Columns.Count)
tbl.Interior.Color = RGB(255, 0, 0)
Set tbl = Range("A1").CurrentRegion
Set tbl = tbl.Offset(1, 0).Resize(tbl.Rows.Count – 1, tbl.Columns.Count)
tbl.Interior.Color = RGB(255, 255, 0)
```

Автоматическое изменение размеров шрифта с тем, чтобы текст умещался в диапазоне

Свойство `ShrinkToFit` объекта `Range`, принимающее логические значения, устанавливает, надо ли автоматически изменять шрифт так, чтобы текст целиком умещался в диапазоне.

Установка стандартной высоты строки и ширины столбца

Свойства `UseStandardHeight` и `UseStandardWidth` объекта `Range` возвращают значение `True`, если все входящие в диапазон строки и столбцы имеют стан-

дартные размеры. Например, следующая инструкция устанавливает стандартную высоту для первой строки.

```
Rows(1).UseStandardHeight = True
```

Автоматический перенос слов

Свойство `WrapText` объекта `Range` принимает логические значения, и если его значение равно `True`, то в диапазоне разрешен перенос текста. В следующем примере в ячейку **B2** вводится текст `Длинный текст`, и в этой ячейке устанавливается режим ввода текста с переносом.

```
With Range("B2")
   .Value = "Длинный текст"
   .WrapText = True
End With
```

Как узнать, на каком листе находится данная активная ячейка

Свойство `Worksheet` объекта `Range` возвращает объект `Worksheet`, в котором находится данный диапазон. Поэтому следующая инструкция возвращает имя листа, на котором расположена активная ячейка.

```
MsgBox ActiveCell.Worksheet.Name
```

Методы объекта *Range*

Объект `Range` обладает большой коллекцией методов, предоставляющих в распоряжение разработчика возможность программировать целый спектр действий от копирования диапазона в буфер обмена до нахождения корня нелинейного уравнения. В последующих разделах подробно на детальных примерах разобраны следующие методы объекта `Range`:

`Activate`	`ClearNotes`	`Find`
`AddComment`	`Copy`	`FindNext`
`AutoFill`	`CopyPicture`	`FindPrevious`
`AutoFit`	`Cut`	`FunctionWizard`
`BorderAround`	`DataSeries`	`GoalSeek`
`Clear`	`Delete`	`Insert`
`ClearComments`	`FillDown`	`PasteSpecial`
`ClearComments`	`FillLeft`	`Replace`
`ClearContents`	`FillRight`	`Select`
`ClearFormats`	`FillUp`	`Show`

Кроме того, методы `Consolidate` подробно рассмотрены в *главе 10*.

Активизация и выбор диапазона

Метод `Activate` объекта `Range` активизирует диапазон, а метод `Select` выбирает диапазон, т. е. возвращает объект `Selection`. Например, в следующем коде сначала активизируется ячейка **A2**, затем в активную ячейку вводится число `1`, после чего выбирается диапазон **A3:A4** и в выбранный диапазон вводится число `3`.

```
Range("A2").Activate
ActiveCell.Value =  1
Range("A3:A4").Select
Selection.Value = 3
```

Вставка и удаление комментариев в диапазон

Метод `AddComment` объекта `Range` вставляет комментарий в диапазон, а метод `ClearComments` его удаляет. Например, следующий код (листинг 3.79) сканирует диапазон **A1:A100** и в случае, если в очередную ячейку входит строка `BHV`, сначала эта ячейка очищается от комментариев, а затем в нее вставляется комментарий `Книга издательства BHV`.

Листинг 3.79. Вставка и удаление комментариев в диапазон

```
Sub DemoComments()
   Dim c As Range
   For Each c In Range("A1:A100")
      If c.Value Like "*BHV*" Then
         c.ClearComments
         c.AddComment "Книга издательства BHV"
      End If
   Next
End Sub
```

Автоматический подбор размеров диапазона так, чтобы в нем помещались введенные данные

Метод `AutoFit` объекта `Range` автоматически подбирает ширину столбца и высоту строки так, чтобы в ней помещались введенные данные. В следующем коде (листинг 3.80) продемонстрирована техника использования метода `AutoFit` при создании заголовка некоей отчетной таблицы.

Листинг 3.80. Автоматический подбор размеров диапазона

```
Sub DemoAutoFit()
   Range("A1").Value = "Июнь"
```

```
    Range("B1").Value = " Июль "
    Range("C1").Value = "Август"
    Columns("A:C").AutoFit
    Range("D1").Value = "Суммарный объем продаж"
    Range("D1").Columns.AutoFit
End Sub
```

Заполнение диапазона прогрессией

Метод `DataSeries` объекта `Range` заполняет диапазон прогрессией. Метод `DataSeries` программирует выполнение команды **Правка | Заполнить | Прогрессия**.

```
DataSeries(RowCol, Type, Date, Step, Stop, Trend)
```

- *`RowCol`* — необязательный параметр, определяющий направление, в котором создается прогрессия. Допустимые значения: `xlRows` (вдоль строк), `xlColumns` (вдоль столбцов). Если параметр опущен, то для задания направления используются размеры данного диапазона;
- *`Type`* — необязательный параметр, специфицирующий тип прогрессии. Допустимыми значениями являются следующие константы `XlDataSeriesType`: `xlDataSeriesLinear` (линейная, используется по умолчанию), `xlGrowth` (геометрическая), `xlChronological` (даты), `xlAutoFill` (автозаполнение);
- *`Date`* — необязательный параметр, задающий тип последовательности дат, если параметр *`Type`* принимает значение `xlChronological`. Допустимыми значениями являются следующие константы `XlDataSeriesDate`: `xlDay` (дни, используется по умолчанию), `xlWeekday` (дни недели), `xlMonth` (месяцы), `xlYear` (годы);
- *`Step`* — необязательный параметр, определяющий шаг изменения прогрессии. По умолчанию полагается равным 1;
- *`Stop`* — необязательный параметр, задающий предельное значение прогрессии. По умолчанию строится прогрессия во всем выделенном диапазоне;
- *`Trend`* — необязательный параметр, принимает логические значения. Если значение параметра равно `True`, то создается арифметическая или геометрическая прогрессия, а если `False`, то создается список.

Например, следующие инструкции (листинг 3.81, *а*) выведут членов арифметической прогрессии, шаг которой равен 2, 0 является ее первым членом, а 10 — последним, т. е. в диапазон **A1:A6** будут выведены следующие числа: 0, 2, 4, 6, 8 и 10 (рис. 3.17).

Листинг 3.81, *а*. Прогрессия

```
Range("A1").Value = 0
Range("A1").DataSeries Rowcol:=xlColumns, _
                       Type:=xlDataSeriesLinear, _
                       Step:=2, _
                       Stop:=10
```

Следующие инструкции (листинг 3.81, *б*) введут в диапазон **B1:B5** членов геометрической прогрессии с коэффициентом — 3, первый член которой — 1. Таким образом, в диапазон **B1:B5** будут выведены значения 1, 3, 9, 27 и 81 (рис. 3.17).

Листинг 3.81, *б*. Прогрессия

```
Range("B1").Value = 1
Range("B1:B5").DataSeries Rowcol:=xlColumns, _
                       Type:=xlGrowth, _
                       Step:=3
```

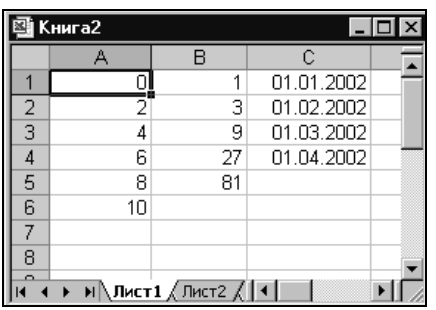


Рис. 3.17. Прогрессии

Следующие инструкции (листинг 3.81, *в*) выведут в диапазон **C1:C4** последовательность дат, члены которой разнятся ровно на месяц, т. е. значения `01.01.2002`, `01.02.2002`, `01.03.2002`, `01.04.2002` (рис. 3.17).

Листинг 3.81, *в*. Прогрессия

```
Range("C1").Value = "1/01/02"
Range("C1:C4").DataSeries Rowcol:=xlColumns, _
                          Type:=xlChronological, _
                          Date:=xlMonth
```

Автозаполнение ячеек диапазона элементами последовательности

Метод `AutoFill` объекта `Range` производит автозаполнение ячеек диапазона элементами последовательности. Метод `AutoFill` отличается от метода `DataSeries` тем, что явно указывается диапазон, в котором будет располагаться прогрессия. Метод `AutoFill` программирует выполнение копирования данных на диапазон, когда пользователь располагает указатель мыши на маркере заполнения исходного диапазона и перемещает его вниз и вправо, выделяя весь диапазон, в который переносятся исходные данные.

expression.AutoFill(*Destination*, *Type*)

- *expression* — обязательный элемент, который задает диапазон, с которого начинается заполнение;
- *Destination* — обязательный параметр, определяющий диапазон, который заполняется. Этот диапазон должен содержать диапазон указанный в *expression*;
- *Type* — необязательный параметр, указывающий тип заполнения. Допустимыми значениями являются следующие константы `XlAutoFillType`: `xlFillDefault`, `xlFillSeries`, `xlFillCopy`, `xlFillFormats`, `xlFillValues`, `xlFillDays`, `xlFillWeekdays`, `xlFillMonths`, `xlFillYears`, `xlLinearTrend`, `xlGrowthTrend`. По умолчанию используется тот тип заполнения, который наиболее подходит к данным из диапазона, указанного в *expression*.

Например, следующие инструкции (листинг 3.82, *а*) заполняют диапазон **A1:A5** членами арифметической прогрессии, причем ее первыми двумя членами являются 1 и 3, т. е. те значения, которые предварительно были введены в ячейки **A1** и **A2** (рис. 3.18).

Листинг 3.82, *а*. Последовательности

```
Range("A1").Value = 1
Range("A2").Value = 3
Range("A1:A2").AutoFill Destination:=Range("A1:A5"), Type:=xlLinearTrend
```



Рис. 3.18. Последовательности

Следующие инструкции (листинг 3.82, *б*) выводят несколько членов геометрической прогрессии с теми же двумя начальными значениями в диапазон **B1:B5** (см. рис. 3.18).

Листинг 3.82, *б*. Последовательности

```
Range("B1").Value = 1
Range("B2").Value = 3
Range("B1:B2").AutoFill Destination:=Range("B1:B5"), Type:=xlGrowthTrend
```

Следующие инструкции (листинг 3.82, *в*) выведут в диапазон **C1:C3** последовательность значений `Лето 2000`, `Лето 2001` и `Лето 2002` с шагом 1, определенным по умолчанию методом `AutoFill` (см. рис. 3.18).

Листинг 3.82, *в*. Последовательности

```
Range("C1").Value = "Лето 2000"
Range("C1").AutoFill Destination:=Range("C1:C3"), Type:=xlFillSeries
```

Приведенные ниже две инструкции (листинг 3.82, *г*) выводят в диапазон **D1:D3** первые три члена списка — имена месяцев, начиная с `Январь` (см. рис. 3.18).

Листинг 3.82, *г*. Последовательности

```
Range("D1").Value = "Январь"
Range("D1").AutoFill Destination:=Range("D1:D3"), Type:=xlFillSeries
```

Следующие инструкции (листинг 3.82, *д*) копируют содержимое ячейки **E1** на все ячейки диапазона **E1:E3** (см. рис. 3.18).

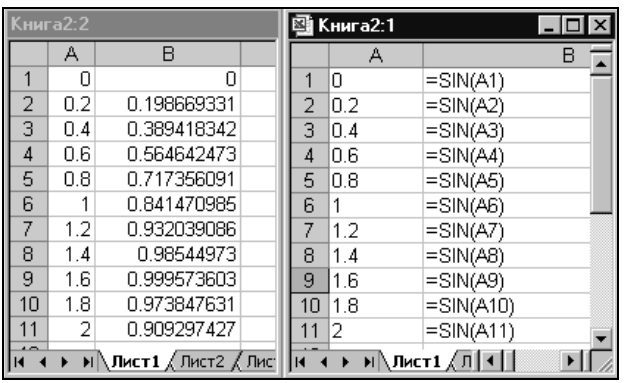
Листинг 3.82, *д*. Последовательности

```
Range("E1").Value = "Январь"
Range("E1").AutoFill Destination:=Range("E1:E3"), Type:=xlCopy
```

Табуляция функции

Метод `AutoFill` позволяет решить задачу табуляции функции, т. е. вывода ее значений при изменении значения ее параметра. Например, требуется найти значения функции Sin(*x*) при значении параметра *x*, изменяющегося от 0 до 2 с шагом 0,2. Это можно сделать следующим образом (листинг 3.83). Сначала в ячейку **A1** ввести первый член арифметической последователь-

ности требуемых значений параметра, а потом с помощью метода `DataSeries` построить и всю последовательность вдоль столбца **A**. Затем определить как текущий диапазон с этими значениями. Диапазон, в котором расположатся соответствующие значению функции, разместится в столбце **B**, и его можно найти при помощи свойства `Offset`. Далее остается самая малость. В ячейку **B1** ввести формулу `=SIN(A1)` для нахождения значения функции при значении параметра, равного 0, а затем скопировать данную формулу на весь диапазон, отводимый под искомые значения функции (рис. 3.19).



	A	B
1	0	0
2	0.2	0.198669331
3	0.4	0.389418342
4	0.6	0.564642473
5	0.8	0.717356091
6	1	0.841470985
7	1.2	0.932039086
8	1.4	0.98544973
9	1.6	0.999573603
10	1.8	0.973847631
11	2	0.909297427

	A	B
1	0	=SIN(A1)
2	0.2	=SIN(A2)
3	0.4	=SIN(A3)
4	0.6	=SIN(A4)
5	0.8	=SIN(A5)
6	1	=SIN(A6)
7	1.2	=SIN(A7)
8	1.4	=SIN(A8)
9	1.6	=SIN(A9)
10	1.8	=SIN(A10)
11	2	=SIN(A11)

Рис. 3.19. Табуляция функции

Листинг 3.83. Табуляция функции

```
Sub DemoDataSeries()
   Range("A1").Value = 0
   Range("A1").DataSeries Rowcol:=xlColumns, _
                          Type:=xlDataSeriesLinear, _
                          Step:=0.2, _
                          Stop:=2
   Dim rgn As Range
   Set rgn = Range("A1").CurrentRegion
   Set rgn = rgn.Offset(0, 1)
   Range("B1").Formula = "=SIN(A1)"
   Range("B1").AutoFill Destination:=rgn, Type:=xlCopy
End Sub
```

Заполнение диапазона по одному значению

Метод `FillDown` объекта `Range` заполняет сверху вниз диапазон на основе значений, расположенных в верхней строчке этого диапазона, причем данные значения перекопируются во все остальные ячейки диапазона.

Метод `FillUp` объекта `Range` заполняет снизу вверх диапазон на основе значений, расположенных в нижней строчке этого диапазона.

Метод `FillLeft` объекта `Range` заполняет справа налево диапазон на основе значений, расположенных в крайнем правом столбце этого диапазона.

Метод `FillRight` объекта `Range` заполняет слева направо диапазон на основе значений, расположенных в крайнем левом столбце этого диапазона.

Например, в данной инструкции содержание ячейки **A10** копируется в каждую из ячеек диапазона **A1:A9**.

```
Range("A1:A10").FillUp
```

Обрамление диапазона границей

Метод `BorderAround` объекта `Range` обрамляет диапазон границей.

```
BorderAround(LineStyle, Weight, ColorIndex, Color)
```

- *`LineStyle`* — необязательный параметр, задающий стиль границы. Допустимыми значениями являются следующие константы `XlLineStyle`: `xlContinuous`, `xlDash`, `xlDashDot`, `xlDashDotDot`, `xlDot`, `xlDouble`, `xlLineStlyeNone`, `xlSlantDashDot` и `xlLineStlyeNone`;
- *`Weight`* — необязательный параметр, определяющий толщину границы. Допустимыми значениями являются следующие константы `XlBorderWeight`: `xlHairline`, `xlMedium`, `xlThick` и `xlThin`;
- *`ColorIndex`* — необязательный параметр, задающий цвет в соответствии с текущей палитрой цветов. Также допустимы следующие константы `XlColorIndex`: `xlColorIndexAutomatic` и `xlColorIndexNone`;
- *`Color`* — необязательный параметр, определяющий цвет в соответствии с RGB-моделью.

Например, следующая инструкция создает вокруг диапазона **A1:B2** двойную толстую границу зеленого цвета.

```
Range("A1:B2").BorderAround LineStyle:=xlDouble, _
                            Weight:=xlThick, _
                            Color:=RGB(0, 255, 0)
```

Очиста ячейки

Методы `Clear`, `ClearComments`, `ClearContents`, `ClearFormats` и `ClearNotes` объекта `Range` очищают диапазон и в диапазоне комментарии, содержание, форматы и примечания. Например, следующая инструкция очищает диапазон **A1:G37**.

```
Range("A1:G37").Clear
```

Копирование, вырезание и удаление данных из диапазона

Метод `Copy` объекта `Range` копирует диапазон в другой диапазон или в буфер обмена.

```
Copy(Destination)
```

Здесь *Destination* — необязательный параметр, определяющий диапазон, в который копируется данный диапазон. Если этот параметр опущен, то копирование происходит в буфер обмена. В данном примере диапазон **A1:D4** активного рабочего листа копируется в диапазон **E5:H8** рабочего листа **Лист2**.

```
Range("A1:D4").Copy Worksheets("Лист2").Range("E5")
```

Метод `Cut` объекта `Range` вырезает, т. е. копирует с удалением диапазон в указанный диапазон или в буфер обмена.

```
Cut(Destination)
```

Здесь *Destination* — необязательный параметр, задающий диапазон, в который копируется данный диапазон. Если этот параметр опущен, то диапазон копируется в буфер обмена. В данном примере диапазон **A1:D4** рабочего листа **Лист1** копируется с удалением в буфер обмена.

```
Worksheets("Лист1").Range("A1:D4").Cut
```

Метод `Delete` объекта `Range` удаляет диапазон. В данном примере удаляется третья строка активной рабочей страницы.

```
Rows(3).Delete
```

Специальная вставка

Метод `PasteSpecial` объекта `Range` производит специальную вставку (special paste) из буфера обмена. Этот метод программирует выполнение на рабочем листе команды **Правка | Специальная вставка**.

```
expression.PasteSpecial(Paste, Operation, SkipBlanks, Transpose)
```

- *expression* — объект типа `Range`, задающий диапазон или ссылку на верхнюю левую ячейку диапазона, в который вставляются данные из буфера обмена;
- *Paste* — необязательный параметр. Определяет ту часть содержания буфера обмена, которая должна быть вставлена в диапазон. Допустимыми значениями являются следующие константы `XlPasteType`: `xlPasteAll`, `xlPasteAllExceptBorders`, `xlPasteColumnWidths`, `xlPasteComments`, `xlPasteFormats`, `xlPasteFormulas`, `xlPasteFormulasAndNumberFormats`, `xlPasteValidation`, `xlPasteValues` и `xlPasteValuesAndNumberFormats`;

- *Operation* — необязательный параметр. Определяет операцию над вставляемыми данными и теми, которые содержатся в диапазоне. Допустимыми значениями являются следующие константы XlPasteSpecialOperation: xlPasteSpecialOperationAdd, xlPasteSpecialOperationDivide, xlPasteSpecialOperationMultiply, xlPasteSpecialOperationNone и xlPasteSpecialOperationSubtract;
- *SkipBlanks* — необязательный параметр. Принимает логические значения и устанавливает, учитываются ли пустые ячейки при вставке;
- *Transpose* — необязательный параметр. Принимает логические значения и устанавливает, вставляется ли диапазон транспонированным.

В приведенном ниже коде данные из диапазона **C1:C5** рабочего листа **Лист1** вставляются в диапазон **D1:D5** того же листа, причем они не заменяют уже существующие данные, а прибавляют к ним данные из диапазона **C1:C5**.

```
With Worksheets("Лист1")
    .Range("C1:C5").Copy
    .Range("D1:D5").PasteSpecial Operation:=xlPasteSpecialOperationAdd
End With
```

Тот же результат можно получить при помощи следующих инструкций, где метод применяется не ко всему диапазону, а только к его верхней левой ячейке.

```
With Worksheets("Лист1")
    .Range("C1:C5").Copy
    .Range("D1").PasteSpecial Operation:=xlPasteSpecialOperationAdd
End With
```

Следующий код копирует данные из диапазона **A1:C1** в буфер обмена и затем вставляет только значения в диапазон **A5:C5**.

```
Range("A1:C1").Copy
Range("A5").PasteSpecial Paste:=xlPasteValues, Operation:=xlNone
```

Вставка диапазона с транспонированием

Вставка диапазона с транспонированием осуществляется методом PasteSpecial при значении его параметра Transpose равным True. Например, в следующем коде (листинг 3.84) значения из диапазона **A1:C2** копируются с транспонированием в диапазон **D1:E3**.

Листинг 3.84. Вставка диапазона с транспонированием

```
Range("A1:C2").Copy
Range("D1").PasteSpecial Paste:=xlPasteAll, _
                         Operation:=xlNone, _
                         Transpose:=True
```

Снятие выделения после специальной вставки

После копирования или вставки данных в буфер обмена исходный диапазон получает выделение, и оно не исчезает после проведения специальной вставки. Для того чтобы это выделение все же удалить, надо установить свойство `CutCopyMode` объекта `Application` равным значению `False`, как показано в следующем коде (листинг 3.85).

Листинг 3.85. Снятие выделения после специальной вставки

```
Range("C1:C5").Copy
Range("D1").PasteSpecial Operation:=xlPasteSpecialOperationAdd
Application.CutCopyMode = False
```

Добавление ячейки, строки или столбца

Метод `Insert` объекта `Range` добавляет в рабочий лист ячейку, строку или столбец.

```
Insert(Shift, CopyOrigin)
```

- *Shift* — необязательный параметр, специфицирующий способ смещения ячеек. Допустимыми значениями являются следующие константы `XlInsertShiftDirection`: `xlShiftToRight` и `xlShiftDown`;
- *CopyOrigin* — необязательный параметр, который задает правило копирования источника данных.

В следующем примере первая инструкция вставляет новую строку перед четвертой строкой рабочего листа, а вторая — вставляет ячейку слева от ячейки **G9**.

```
Rows(4).Insert
Range("G9").Insert Shift:=xlShiftToRight
```

Копирование диапазона в буфер обмена как растровое изображение

Метод `CopyPicture` объекта `Range` копирует в буфер обмена как растровое изображение содержимое указанного диапазона.

```
CopyPicture(Appearance, Format)
```

- *Appearance* — необязательный параметр, специфицирующий способ копирования диапазона. Допустимыми значениями являются следующие константы XlPictureAppearance: xlPrinter и xlScreen;
- *Format* — необязательный параметр, задающий формат изображения. Допустимыми значениями являются следующие константы XlCopyPictureFormat: xlPrinter и xlPicture.

Например, в следующем коде (листинг 3.86) копируется в буфер обмена как растровое изображение диапазон **A1:C2**, затем он вставляется в лист, но уже как графический объект, левый верхний угол которого совпадает с соответствующим углом ячейки **C7**, после чего это изображение поворачивается на 45° против часовой стрелки.

Листинг 3.86. Копирование диапазона в буфер обмена как растровое изображение

```
Range("A1:C2").CopyPicture
Range("C7").Select
ActiveSheet.Paste
Selection.ShapeRange.IncrementRotation -45
```

Отображение окна мастера функций

Метод FunctionWizard объекта Range вызывает отображение окна мастера функций. Например, следующая процедура обработки события BeforeRightClick объекта Worksheet обеспечивает отображение окна мастера функций при щелчке правой кнопкой на любой ячейке листа (листинг 3.87). Для того чтобы блокировать выполнение действия по умолчанию, связанного в системе с щелчком правой кнопкой, значение параметра Cancel устанавливается равным True.

Листинг 3.87. Отображение окна мастера функций. Модуль рабочего листа

```
Private Sub Worksheet_BeforeRightClick(ByVal Target As Range, _
                                       Cancel As Boolean)
   Cancel = True
   ActiveCell.FunctionWizard
End Sub
```

Поиск значения в диапазоне

Метод Find объекта Range производит поиск специфицированной информации в указанном диапазоне и возвращает ссылку на первую найденную

ячейку, в которой требуемая найдена. В случае необнаружения искомых данных метод возвращает значение Nothing.

```
Find(What, After, LookIn, LookAt, SearchOrder, SearchDirection,
↳MatchCase, MatchByte, SearchFormat)
```

- *What* — обязательный параметр, задающий искомые данные;
- *After* — необязательный параметр, указывающий на ячейку, после которой надо производить поиск;
- *LookIn* — необязательный параметр, специфицирующий, где производить поиск. Допустимыми значениями являются следующие константы XlFindLookIn: xlComments, xlFormulas и xlValues;
- *LookAt* — необязательный параметр, указывающий, как надо искать. Допустимыми значениями являются следующие константы XlLookAt: xlWhole и xlPart;
- *SearchOrder* — необязательный параметр, задающий порядок просмотра диапазона. Допустимыми значениями являются следующие константы XlSearchOrder: xlByRows и xlByColumns;
- *SearchDirection* — необязательный параметр, определяющий направление просмотра. Допустимыми значениями являются следующие константы XlSearchDirection: xlNext и xlPrevious;
- *MatchCase* — необязательный параметр, указывающий, надо ли при поиске учитывать регистр букв;
- *MatchByte* — необязательный параметр, применяется редко;
- *SearchFormat* — необязательный параметр, задающий формат поиска.

Например, следующий код (листинг 3.88) производит поиск значения 17 в диапазоне **A1:A10**. В случае его обнаружения на экране отображается окно с адресом обнаруженной ячейки.

Листинг 3.88. Поиск значения

```
Dim rng As Range
Set rng = Range("A1:A10").Find(What:=17, LookIn:=xlValues)
If Not (rng Is Nothing) Then
   MsgBox rng.Address
Else
   MsgBox "Не найдено значение"
End If
```

Следующий код (листинг 3.89) производит поиск подстроки BHV без учета регистра в диапазоне **A1:A10**. В случае успешного поиска на экране отображается окно со значением свойства Value обнаруженной ячейки.

Листинг 3.89. Поиск подстроки без учета регистра

```
Sub DemoFindNoMatchCase ()
   Dim rng As Range
   Set rng = Range("A1:A10").Find(What:="BHV", LookIn:=xlValues, _
                                  LookAt:=xlPart, MatchCase:=False)
   If Not (rng Is Nothing) Then
      MsgBox rng.Value
   Else
      MsgBox "Не найдено подходящее значение"
   End If
End Sub
```

Повторный поиск и поиск всех значений

Методы `FindNext` и `FindPrevious` объекта `Range` реализуют повторный вызов метода `Find` для продолжения специфицированного поиска. Первый из методов производит поиск следующей ячейки, а второй — поиск предыдущей, удовлетворяющей объявленным критериям поиска.

```
FindNext(After)
FindPrevious(After)
```

Здесь *After* — необязательный параметр, указывающий на ячейку, после которой надо производить поиск.

В качестве примера приведем код (листинг 3.90), который производит поиск подстроки `BHV` без учета регистра в диапазоне **A1:A10**. Все найденные ячейки заливаются желтым цветом.

Листинг 3.90. Нахождение всех вхождений подстроки в данный диапазон

```
Sub DemoFind()
   Dim firstAddress As String
   Dim rng As Range
   Set rng = Range("A1:A10").Find(What:="MS", LookIn:=xlValues, _
                                  LookAt:=xlPart, MatchCase:=False)
   If Not (rng Is Nothing) Then
      firstAddress = rng.Address
      Do
         rng.Interior.Color = RGB(255, 255, 0)
         Set rng = Range("a1:a10").FindNext(rng)
      Loop While Not (rng Is Nothing) And rng.Address <> firstAddress
   End If
End Sub
```

Замена значений

Метод `Replace` объекта `Range` производит замену в указанном диапазоне.

```
Replace(What, Replacement, LookAt, SearchOrder, SearchDirection,
↳ MatchCase, MatchByte, SearchFormat, ReplaceFormat)
```

- *What* — обязательный параметр, задающий строку, которая будет заменяться;
- *Replacement* — обязательный параметр, задающий строку, которой будет производиться замещение;
- *LookAt* — необязательный параметр, указывающий, как надо искать. Допустимыми значениями являются следующие константы `XlLookAt`: `xlWhole` и `xlPart`;
- *SearchOrder* — необязательный параметр, задающий порядок просмотра диапазона. Допустимыми значениями являются следующие константы `XlSearchOrder`: `xlByRows` и `xlByColumns`;
- *SearchDirection* — необязательный параметр, определяющий направление просмотра. Допустимыми значениями являются следующие константы `XlSearchDirection`: `xlNext` и `xlPrevious`;
- *MatchCase* — необязательный параметр, указывающий, надо ли при поиске учитывать регистр букв;
- *MatchByte* — необязательный параметр, применяется редко;
- *SearchFormat* — необязательный параметр, задающий формат поиска;
- *ReplaceFormat* — необязательный параметр, задающий формат замены.

Например, в следующем коде в столбце **A** строка `MS` заменяется строкой `Microsoft`.

```
Columns("A").Replace What:="MS", Replacement:="Microsoft", _
                     SearchOrder:=xlByColumns, MatchCase:=True
```

Подбор параметра и решение уравнения с одним неизвестным

Метод `GoalSeek` объекта `Range` подбирает значение параметра (неизвестной величины), являющееся решением уравнения с одним неизвестным. Предполагается, что уравнение приведено к виду: правая часть уравнения является постоянной, не зависящей от параметра, параметр входит только в левую часть уравнения, например:

$$x^3 - 3x - 5 = 0$$

Метод GoalSeek программирует выполнение команды **Сервис | Подбор параметра**. Этот метод вычисляет корень, используя метод последовательных приближений, результат выполнения которого, вообще говоря, зависит от начального приближения. Поэтому для корректности нахождения корня надо позаботиться о корректном указании этого начального приближения.

expression.GoalSeek(*Goal*, *ChangingCell*)

- *expression* — ячейка, в которую введена формула, являющаяся правой частью решаемого уравнения. В этой формуле роль параметра (неизвестной величины) играет ссылка на ячейку, указанную в аргументе *ChangingCell*;
- *Goal* — обязательный параметр, задающий значение левой части решаемого уравнения, не содержащей параметра;
- *ChangingCell* — обязательный параметр, указывающий ссылку на ячейку, отведенную под параметр (неизвестную величину). Значение, введенное в данную ячейку до активизации метода GoalSeek, рассматривается как начальное приближение к искомому корню. Значение, возвращаемое в эту ячейку после выполнения метода GoalSeek, является найденным приближением к корню.

Точность, с которой находится корень и предельно допустимое число итераций, используемых для нахождения корня, устанавливается свойствами MaxChange и MaxIterations объекта Application. Например, определение корня с точностью до 0,0001 максимум за 1000 итераций устанавливается инструкцией:

```
With Application
   .MaxIterations = 1000
   .MaxChange = 0.0001
End With
```

Метод GoalSeek возвращает значение True, если решение найдено, и значение False — в противном случае.

Например, следующий код (листинг 3.91) ищет корень уравнения $x^3 - 3x - 5 = 0$ при начальном приближении 1 (рис. 3.20).

Листинг 3.91. Решение уравнения

```
Sub DemoGoalSeek()
   Range("A1").Name = "x"
   Range("A1").Value = 1
   Range("B1").Formula = "=x^3-3*x-5"
   If Range("B1").GoalSeek(Goal:=0, ChangingCell:=Range("x")) Then
      MsgBox "Корень: " & Range("A1").Value
```

```
   Else
      MsgBox "Корень не найден"
   End If
End Sub
```

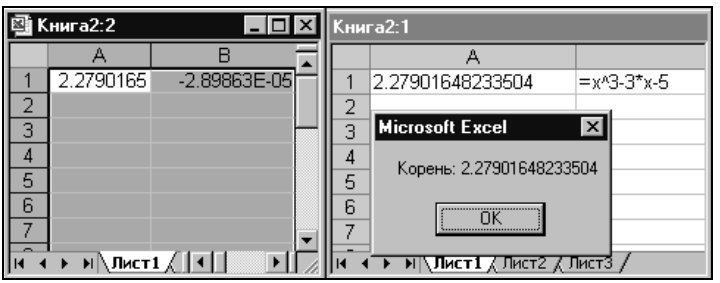


Рис. 3.20. Решение уравнения

Прокрутка экрана с тем, чтобы диапазон отобразился на экране

Метод `Show` объекта `Range` производит прокрутку экрана с тем, чтобы данный диапазон отобразился на экране. Например, следующая инструкция приводит к отображению ячейки **AX100**.

```
Cells(100, 50).Show
```

Семейство *Windows*

Семейство `Windows` представляет набор окон, причем, семейство `Windows`, возвращенное свойством `Windows` объекта `Application`, представляет семейство всех окон данного экземпляра приложения, возвращенное же свойством `Windows` объекта `Workbook` — семейство окон данной книги.

Свойства семейства *Windows*

В семействе `Windows` имеются всего несколько свойств, причем основными из них являются свойства `Count` и `Item`, возвращающие число элементов и указанный элемент семейства. В следующим коде (листинг 3.92) свойства `Count` и `Item` используются для отображения списка имен всех окон данного приложения.

Листинг 3.92. Список имен всех окон

```
Sub DemoWindows()
   Dim i As Integer
```

```
   Dim s As String
   For i = 1 To Windows.Count
      s = s + Windows.Item(i).Caption & vbCr
   Next
   MsgBox s
End sub
```

Упорядочивание окон

В семействе `Windows` имеется единственный метод `Arrange`, который производит упорядочивание окон.

`Arrange(`*`ArrangeStyle`*`, `*`ActiveWorkbook`*`, `*`SyncHorizontal`*`, `*`SyncVertical`*`)`

- *`ArrangeStyle`* — необязательный параметр, определяющий стиль упорядочивания. Допустимыми значениями являются следующие константы `XlArrangeStyle`: `xlArrangeStyleCascade`, `xlArrangeStyleTiled`, `xlArrangeStyleHorizontal`, `xlArrangeStyleVertical`;
- *`ActiveWorkbook`* — необязательный параметр, который при значении `True` упорядочивает только видимые окна активной книги, а при значении `False` — все окна;
- *`SyncHorizontal`* — необязательный параметр, который игнорируется, если значение параметра *`ActiveWorkbook`* равно `False`. Если значение параметра *`SyncHorizontal`* равно `True`, то окна синхронизированы при горизонтальной прокрутке;
- *`SyncVertical`* — необязательный параметр, который игнорируется, если значение параметра *`ActiveWorkbook`* равно `False`. Если значение параметра *`SyncHorizontal`* равно `True`, то окна синхронизированы при вертикальной прокрутке.

Например, следующая инструкция задает каскадное расположение всех окон.

```
Application.Windows.Arrange ArrangeStyle:=xlArrangeStyleCascade
```

Объект *Window*

Объект `Window` инкапсулирует в себе характеристики и параметры данного окна. Посредством его свойств и методов можно управлять, например, видимостью полос прокрутки, сетки и т. д. Он обычно применяется при конструировании форм на основе рабочего листа. Многие из его свойств можно установить вручную на вкладке **Вид** диалогового окна **Параметры**, отображаемого на экране выбором команды **Сервис | Параметры**.

Свойства объекта *Window*

Свойства объекта `Window` позволяют управлять внешним видом окна, например отображением сетки, строки формул. В табл. 3.34 приведены основные свойства этого объекта.

Таблица 3.34. *Основные свойства объекта* `Window`

Свойство	Описание
`ActiveCell`	Активная ячейка активного окна
`ActiveChart`	Активная диаграмма активного окна
`ActiveSheet`	Активный лист активного окна
`Caption`	Заголовок окна
`DisplayFormulas`	Если значение параметра равно `True`, то на рабочем листе отображаются формулы
`DisplayGridlines`	Если значение параметра равно `True`, то на рабочем листе отображается сетка
`DisplayHeadings`	Если значение параметра равно `True`, то на рабочем листе отображаются заголовки строк и столбцов
`DisplayHorizontalSrollBar`	Если значение параметра равно `True`, то отображается горизонтальная полоса прокрутки
`DisplayOutline`	Если значение параметра равно `True`, то отображается структура
`DisplayVerticalScrollBar`	Если значение параметра равно `True`, то отображается вертикальная полоса прокрутки
`DisplayWorkbookTabs`	Если значение параметра равно `True`, то отображаются ярлыки листов
`DisplayZeros`	Если значение параметра равно `True`, то нули отображаются
`EnableResize`	Если значение параметра равно `True`, то разрешено изменение размеров окна, а если `False`, то оно запрещено
`FreezePanes`	Если значение параметра равно `True`, то области закреплены
`GridlineColor`	Задает цвет решетки, используя RGB-модель
`GridlineColorIndex`	Задает цвет решетки, на основе индексов текущей палитры цветов

Таблица 3.34 *(окончание)*

Свойство	Описание
`Height`, `Width`	Высота и ширина окна
`Index`	Индекс объекта `Window` в семействе `Windows`
`Left`, `Top`	Координата левого верхнего угла окна
`OnWindow`	Имя процедуры, которая выполняется при активизации окна
`Panes`	Возвращает семейство `Panes` объектов `Pane`, инкапсулирующих информацию об областях окна
`RangeSelection`	Возвращает объект `Range`, представляющий выбранные ячейки в специфицированном окне
`ScrollColumn`, `ScrollRow`	Возвращают номер первых столбца и строки, отображаемых в окне
`SelectedSheets`	Возвращает семейство всех выбранных листов рабочей книги
`Selection`	Возвращает выбранный объект в окне
`Split`	Возвращает значение `True`, если окно разделено на области
`SplitColumn`, `SplitRow`	Возвращают номера столбца и строки, в которых окно разделено на области
`SplitHorizontal`, `SplitVertical`	Возвращают расстояние в пунктах до точки, в которой окно разделено на области
`TabRatio`	Возвращает отношение ширины области, отводимой под ярлычки листов, к длине горизонтальной полосы прокрутки
`UsableHeight`, `UsableWidth`	Возвращают максимальную высоту и ширину, которые окно может иметь в приложении
`WindowNumber`	Возвращает номер окна для просмотра данной книги. Если она просматривается только в одном окне, то номер равен единице
`WindowState`	Задает состояние окна. Допустимыми значениями являются следующие константы `XlWindowState`: `xlMaximized`, `xlNormal` и `xlMinimized`
`Zoom`	Задает масштаб просмотра окна

Установка местоположения и размеров окна

Свойства `Top`, `Left`, `Height` и `Width` позволяют установить местоположение и размеры окна. Конечно, размеры окна лучше задавать не в абсолютных, а в относительных величинах по отношению к максимально допустимым размерам, возвращаемым свойствами `UsableHeight` и `UsableWidth` объекта `Application`, как это делается в приводимом коде (листинг 3.93).

Листинг 3.93. Установка местоположения и размеров окна. Модуль `ЭтаКнига`

```
Private Sub Workbook_Open()
   With ActiveWindow
      .WindowState = xlNormal
      .Top = 0
      .Left = 0
      .Height = Application.UsableHeight / 2
      .Width = Application.UsableWidth / 2
   End With
End Sub
```

Методы объекта *Window*

Объект `Window` обладает большим количеством методов, предоставляющих широкий спектр действий от активизации конкретного окна до определения объекта, расположенного в точке со специфицированными координатами. В табл. 3.35 перечислены основные методы этого объекта.

Таблица 3.35. *Основные методы объекта* `Window`

Метод	Описание
`Activate`, `ActivateNext`, `ActivatePrevious`	Активизирует данное, следующее и предыдущее окно
`Close`	Закрывает окно
`LargeScroll`	Постраничная прокрутка окна
`NewWindow`	Создает новое окно или копию специфицированного окна
`PointsToScreenPixelsX`, `PointsToScreenPixelsY`	Переводят пункты в пикселы
`PrintOut`	Печать окна
`PrintPreview`	Предварительный просмотр перед печатью

Таблица 3.35 (окончание)

Метод	Описание
RangeFromPoint	Возвращает объект Shape или Range, расположенный в указанных координатах. Если в специфицированном местоположении таких объектов нет, то возвращается значение Nothing
SmallScroll	Прокрутка окна по строчкам и столбцам

Перемещение по иерархии объектов при помощи свойства *Parent*

Свойство Parent возвращает ссылку на объект, стоящий выше по отношению к данному в иерархии MS Excel. Например, у объекта Worksheet таким объектом является Workbook, а у объекта Workbook — объект Application. Следующий код (листинг 3.94) является примером скрипта, т. е. обычного текстового файла с расширением vbs, код в котором написан на языке VBScript, и который может без компиляции выполняться прямо в Windows. В нем при помощи функции GetObject в MS Excel открывается указанный файл. Функция GetObject возвращает объект Workbook, у которого нет свойства Visible, и поэтому пользователь не увидит открытую книгу. Как преодолеть это затруднение? Да очень просто! Надо получить ссылку на объект Application при помощи свойства Parent, а уже у последнего свойство Visible имеется, и, кроме того, требуется установить значение свойства Visible окна, в котором выводится книга, равным также значению True.

Листинг 3.94. Перемещение по иерархии объектов при помощи свойства Parent. Файл DemoParent.vbs

```
Dim wb
Dim fname
fname = "f:\vba2002\pages.xls"
Set wb = GetObject(fname)
wb.Parent.Visible = True
wb.Windows(1).Visible = True
wb.Worksheets(1).Select
Set wb = Nothing
```

Объект *Debug* и окно *Immediate*

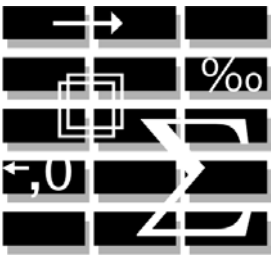
Окно **Immediate** отображается на экране в редакторе Visual Basic выбором команды **View | Immediate Window** или нажатием комбинации клавиш <Ctrl>+<G>. Это окно используется для вывода значений при отладке программ. В окно **Immediate** через объект `Debug` можно выводить значения выражений. У объекта `Debug` имеются только два метода, приведенных в табл. 3.36, и ни одного свойства.

Таблица 3.36. *Методы объекта* `Debug`

Метод	Описание
`Print`	Выводит данные в окно **Immediate**. `Print` *`OutputList`* Здесь *`OutputList`* — необязательный параметр, задающий список выражений, значения которых отображаются в окне **Immediate**. Разделителями в списке выражений могут быть ";" — последовательный вывод и "," — вывод с начала новой зоны

`Assert`	Прерывает выполнение кода в строке с данной инструкцией.
	`Assert` *`BooleanExpression`*
	Здесь *`BooleanExpression`* — обязательный параметр, определяющий, надо ли приостанавливать выполнение кода.
	В следующем примере моделируется бросание игральной кости. Код выполняется до тех пор, пока не выпадет шесть очков.
	`Dim Score As Integer`
	`Dim flg As Boolean`
	`flg = False`
	`Randomize`
	`Do`
	`  Score = Int(6 * Rnd()) + 1`
	`  If Score = 6 Then`
	`  flg = False`
	`  Else`
	`  Debug.Print Score`
	`  flg = True`
	`  End If`
	`  Debug.Assert flg`
	`Loop`

ГЛАВА 4



Основы программирования на VBA

В предыдущих двух главах мы познакомились с основными объектами VBA, технологией разработки приложений и даже создали несколько приложений. При создании приложений, конечно, были использованы конструкции и функции языка VBA. Прежде чем двигаться дальше по пути разработки приложений, остановимся на несколько мгновений, чтобы перевести дыхание и познакомиться с основами языка VBA.

Допустимые имена

В VBA пользователь определяет имена переменных, функций, процедур, типов, констант и других объектов. Вводимые пользователем имена должны отражать суть обозначаемого объекта так, чтобы делать программу легко читаемой. В VBA имеются следующие ограничения на имена:

- длина имени не должна превышать 255 символов;
- имя не может содержать точек, пробелов и следующих символов: %, &, !, #, @, $;
- имя может содержать любую комбинацию букв, цифр и символов, начинающуюся с буквы;
- имена должны быть уникальны внутри области, в которой они определены;
- не следует использовать имена, совпадающие с ключевыми словами VBA и именами встроенных функций и процедур.

Хотя верхний или нижний регистр букв в имени не имеет значения, умелое использование его может существенно облегчить понимание содержательной стороны переменной.

Например, вместо плоских и невыразительных имен

```
firstname
x_initialvalue
```

предпочтительнее использовать имена, которые легко читаются с одного взгляда, благодаря акцентированию входящих в них компонентов разумным использованием верхнего регистра букв:

```
FirstName
x_InitialValue
```

Имена переменных должны быть описательными — для простоты чтения кода — и, желательно, короткими — для быстроты набора кода. Иногда в именах переменных используются префиксы, указывающие тип и область действия переменной, например, `gintScore` — для обозначения глобальной переменной целого типа.

Типы данных

Типы данных относятся к самим фундаментальным понятиям любого языка. Тип данных определяет множество допустимых значений, которое может принимать указанная переменная. В табл. 4.1 перечислены типы данных, используемых в VBA.

Таблица 4.1. *Типы данных, используемые в VBA*

Тип данных	Размер (байт)	Диапазон значений
`Byte` (байт)	1	От 0 до 255
`Boolean` (логический)	2	`True` или `False`
`Integer` (целое)	2	От –32 768 до 32 767
`Long` (длинное целое)	4	От –2 147 483 648 до 2 147 483 647
`Single` (с плавающей точкой обычной точности)	4	От –3,402823E38 до –1,401298E–45 для отрицательных значений; от 1,401298E–45 до 3,402823E38 для положительных значений
`Double` (с плавающей точкой двойной точности)	8	От –1,79769313486232E308 до –4,94065645841247E–324 для отрицательных значений; от 4,94065645841247E–324 до 1,79769313486232E308 для положительных значений

Таблица 4.1 *(окончание)*

Тип данных	Размер (байт)	Диапазон значений
`Currency` (денежный)	8	От –922 337 203 685 477,5808 до 922 337 203 685 477,5807
`Decimal` (масштабируемое целое)	14	+/–79228162514264337593543950335 с 28 знаками справа от запятой; минимальное ненулевое значение имеет вид +/–0,0000000000000000000000000001
`Date` (даты и время)	8	От 1 января 100 г. до 31 декабря 9999 г.
`Object` (объект)	4	Любой указатель объекта
`String` (строка переменной длины)	10 + длина строки	От 0 до приблизительно 2 миллиардов
`String` (строка постоянной длины)	Длина строки	От 1 до приблизительно 65 400
`Variant` (числовые подтипы)	16	Любое числовое значение вплоть до границ диапазона для типа `Double`
`Variant` (строковые подтипы)	22 + длина строки	Как для строки (`String`) переменной длины
Тип данных, определяемый пользователем (с помощью ключевого слова `Type`)	Объем определяется элементами	Диапазон каждого элемента определяется его типом данных

Переменные

В VBA *переменная* (variable) используется для временного хранения данных в оперативной памяти, т. е. она идентифицирует область в памяти, где хранится некоторая информация. После объявления переменная указывает на одну и ту же область памяти до тех пор, пока не будет уничтожена. Программист не должен указывать явное место хранение данных, ему для ссылки на область памяти достаточно лишь указать на имя.

Уведомить VBA о существовании переменной, или *объявить* переменную можно двумя способами: *явным и неявным*.

Явное объявление означает, что переменная должна быть объявлена прежде, чем ее можно использовать, и оно производится при помощи операторов

Dim, Private, Static или Public, которые также определяют и область видимости переменной:

```
Dim | Static | Private | Public varname1 [As type1][, varname2 [As type2]]...
```

- *varname* — имя переменной, удовлетворяющее стандартным правилам именования переменных;
- *type* — тип данных переменной. Для каждой описываемой переменной следует использовать отдельное предложение As *type*.

Явное объявление каждой переменной делает программу надежнее и, кроме того, убыстряет ее работу, т. к. VBA не требуется тратить время на распознавание типа неописанной переменной при каждом обращении к ней.

Например, следующая инструкция объявляет переменную типа Integer.

```
Dim N As Integer
```

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

```
Dim x
```

Хотя переменные обычно объявляются в начале процедуры или функции, они могут быть объявлены в любом месте до того, как в коде встречается первая ссылка на них. Например:

```
Dim x As Single
x = 1
Dim y As Single
y = x + 1
```

Примечание

Довольно распространенной ошибкой является использование следующей инструкции:

```
Dim a, b As Single
```

в предположении, что обе переменные будут иметь тип Single. На самом деле, только переменная b будет иметь тип Single, в то время как переменная a — тип Variant.

Неявное объявление переменной осуществляется включением в конец его имени специального символа, устанавливающего тип переменной. В этом случае переменную не надо объявлять перед тем, как ее применять. Для установки типов допустимо использование специальных символов, приведенных в табл. 4.2.

Таблица 4.2. *Специальные символы для установки типов*

Специальный символ	Тип	Специальный символ	Тип
%	Integer	#	Double
&	Long	@	Currency
!	Single	$	String

Директива *Option Explicit*

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

Область видимости переменной

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

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

Время жизни переменной

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

Ключевое слово *Null*

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

Комментарии

Текст, следующий в программе за символом (') вплоть до конца строки, игнорируется компилятором и представляет собой *комментарий*.

Комментарии позволяют добавлять описания и пояснения для тех программистов, которые в будущем будут разбираться в вашей программе. Вполне возможно, что на их месте спустя несколько месяцев или лет вы окажетесь сами. Комментируя всю программу, вы экономите время. Кроме того, при написании комментариев нередко в программе обнаруживаются ошибки. Когда приходится объяснять кому-то, что происходит, то поневоле задумаешься над этим сам.

Комментарии также полезны при отладке программ. Они позволяют временно отключать строки кода программы при ее отладке.

Приведем возможные варианты использования комментариев в коде программы.

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

Перенос строки кода

Расположение символов <Пробел>+<Знак подчеркивания> в конце строки обеспечивает то, что последующая строка является продолжением предыдущей. При этом надо помнить, что:

- нельзя разбивать переносом строковые константы;
- допустимо не более семи продолжений одной и той же строки;
- сама строка не может состоять более чем из 1024 символов.

В следующем примере первая из конструкций является разбиением второй на две строки:

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

и

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

Для переноса строковой константы ее надо представить как результат конкатенации нескольких строковых констант, и перенос строки производить по операции конкатенации.

Приведем пример корректного и некорректного переноса строки "`Visual Basic for Applications`":

Некорректный перенос:

```
s = "Visual Basic for _
     Applications"
```

Корректный перенос:

```
s = "Visual Basic " _
  & "for Applications"
```

Расположение нескольких операторов в одной строке

Использование знака двоеточия позволяет разместить несколько операторов на одной строке. Таким образом, следующие две конструкции эквивалентны:

```
x = 1     ' в переменной x содержится 1
x = x + 2 ' в переменной x содержится 3
```

и

```
x = 1 : x = x + 2
```

Строки

Строка (`string`) представляет собой последовательность символов, каждый из которых имеет тип `Char`.

Последовательность символов, присваиваемая строковой переменной, должна быть окружена кавычками.

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

Строковая переменная может быть объявлена как строка переменной длины и как строка постоянной длины. В следующем примере переменная `LastName` объявлена как строка переменной длины, а переменная `State` — как строка постоянной длины, состоящая из трех литер. Если переменной `State` будет присвоено значение какого-то строкового выражения, состоящего более чем из двух литер, лишние справа литеры при присваивании будут обрезаны.

```
Dim LastName As String
Dim State As String * 2
```

Конкатенация строк

В VBA имеется единственная строковая операция — *конкатенация*. Эта операция применяется для объединения нескольких строк в одну. Операция конкатенация обозначается символом амперсанда "&" или сложения "+". Во избежание путаницы, как правило, применяется операция конкатенации со знаком "&". При объединении двух строк вторая строка добавляется непосредственно в конец первой. Результатом является строка большего размера, содержащая целиком обе исходные строки.

В следующем примере переменной `s` присваивается значение "`Visual Basic for Applications`".

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

Ключевое слово *Empty* и пустая строка

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

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

Массивы

Как и в других языках программирования, в VBA вы можете использовать массивы. Массив, как и любую переменную, надо объявлять, используя операторы `Dim`, `Static`, `Private` и `Public`, которые также задают область видимости переменной.

```
Dim | Static | Private | Public VarName(Indexes) [As Type]
↳[, VarName (Indexes) [As Type]] ...
```

- *VarName* — имя переменной, удовлетворяющее стандартным правилам именования переменных;
- *Indexes* — размерности массива; допускается описание до 60 размерностей. При определении этого параметра надо указывать верхнюю, а также нижнюю границу, если последняя не подразумевается равной той, которая установлена по умолчанию:

  ```
  [LowBound To] UpperBound[, [LowBound To] UpperBound] ...
  ```

  Если нижний индекс не задан явно, нижняя граница массива определяется директивой `Option Base`. Если отсутствует директива `Option Base`, нижняя граница массива равняется нулю;
- *Type* — тип данных массива.

Например, в следующем операторе объявляется одномерный массив (вектор) из 12 целых чисел, причем по умолчанию первый элемент массива — `A(0)`, а последний — `A(11)`. В этом случае говорят, что 0 — базовый индекс.

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

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

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

Изменение базового индекса

Можно изменить базовый индекс, написав в области объявлений модуля директиву `Option Base 1`. После этого индексы массивов `A` и `B` будут начинаться с единицы.

Например, в следующем операторе объявляется вектор, состоящий из 11 элементов.

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

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

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

Поэлементная инициализация массива

Поэлементную инициализацию массива можно призводить:

- последовательностью операторов:

```
Dim B(1, 1) As Single
B(0, 0) = 2 : B(0, 1) = 4
B(1, 0) = 1 : B(1, 1) = 6
Dim M(1 To 9, 1 To 9) As Integer
```

- оператором цикла:

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

Инициализация массива при помощи функции *Array*

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

- Инициализация одномерного массива:

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

- Инициализация многомерного массива:

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

Массив и диапазон

В VBA имеется тесная связь между диапазонами и массивами. Допустимо как заполнение массива одним оператором присваивания значениями из ячеек диапазона, так и наоборот, заполнение диапазона ячеек одним оператором присваивания элементами массива. При этом массив должен быть объявлен как переменная типа `Variant`.

- Инициализация массива из диапазона одним оператором присваивания:

```
Dim r As Range
Set r = Range("C1:D3")
Dim M As Variant
M = r.Value

Dim i As Integer
Dim j As Integer
For i = 1 To r.Rows.Count
   For j = 1 To r.Columns.Count
      Cells(i, j).Value = M(i, j)
   Next
Next
```

- Заполнение диапазона из массива посредством одного оператора присваивания:

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

Dim r As Range
Set r = Range(Cells(1, 1), Cells(9, 9))
r.Value = M
```

Динамические массивы

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

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

Затем в программе следует вычислить необходимый размер массива в некоторой переменной, например *n*, и изменить размер динамического массива с помощью инструкции ReDim.

```
ReDim [Preserve] varname(indexes) [As type][, varname(indexes)
↳ [As type]] ...
```

- *Preserve* — ключевое слово, используемое для сохранения данных в существующем массиве при изменении значения последней размерности. Если ключевое слово *Preserve* не используется, то данные в массиве при изменении размерности не сохраняются;
- *varname* — имя переменной, удовлетворяющее стандартным правилам именования переменных;
- *indexes* — размерности переменной массива; допускается описание до 60 размерностей. Если нижний индекс не задан явно, нижняя граница массива определяется инструкцией Option Base. Если отсутствует инструкция Option Base, нижняя граница массива равняется нулю;
- *type* — тип данных массива.

В следующем примере сначала объявляется динамический массив, а затем устанавливаются границы его индекса.

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

Допустимо повторное использование инструкции ReDim для изменения числа элементов и размерностей массива. В следующем примере (листинг 4.1) создается массив с результатами бросания монеты. Монета бросается до тех пор, пока три раза не выпадет орел. Размерность динамического массива после каждого броска корректируется с сохранением в нем ранее записанных результатов бросания монеты, за счет использования ключевого слова *Preserve*.

Листинг 4.1. Изменение размерности динамического массива с сохранением содержания

```
Dim Attempt()
Dim i As Integer
Dim score As Integer
Dim coin As Integer
i = 0
score = 0
```

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

Определение границ индексов массива

Функции `LBound`, `UBound` возвращают минимальное и максимальное допустимые значения указанного индекса массива.

```
LBound(arrayname[, dimension])
UBound(arrayname[, dimension])
```

- *arrayname* — обязательный параметр, задающий имя переменной массива;
- *dimension* — необязательный параметр, указывающий размерность, нижнюю или верхнюю границу которой возвращает функция. Для первой размерности следует указать 1, для второй 2 и т. д. Если параметр опущен, то подразумевается значение 1.

Например, в следующем коде отобразится `100` и `5`.

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

Следующие инструкции позволяют перебирать элементы массива без указания его размерности.

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

Как проверить, содержит ли переменная типа *Variant* массив значений

Функция `IsArray` возвращает значение `True`, если указанная переменная типа `Variant` содержит массив, и значение `False` — в противном случае. Например, следующая процедура (листинг 4.2), обрабатывающая событие

SelectionChange объекта Worksheet, отображает сообщение, если в выделенном диапазоне имеется массив значений.

Листинг 4.2. Как проверить, содержит ли переменная типа Variant массив значений. Модуль рабочего листа

```
Private Sub Worksheet_SelectionChange(ByVal Target As Range)
   Dim A As Variant
   A = Target.Value
   If IsArray(A) Then
      MsgBox "Содержит массив"
   Else
     MsgBox "Не содержит массив"
   End If
End Sub
```

Повторная инициализация массива и высвобождение памяти, выделенной под массив

Оператор Erase повторно инициализирует элементы массивов фиксированной длины и освобождает память, отведенную для динамического массива. Оператор Erase устанавливает элементы массивов фиксированной длины следующим образом:

- массив чисел или строк фиксированной длины (присваивает каждому элементу значение 0);
- массив строк переменной длины (присваивает каждому элементу значение пустой строки);
- массив типа Variant (присваивает каждому элементу значение Empty).

Например, в следующем коде отобразится сначала 1, а затем 0.

```
Dim A(2) As Integer
A(2) = 1
MsgBox A(2)
Erase A
MsgBox A(2)
```

Оператор Erase также освобождает память, используемую динамическими массивами. Перед тем как из программы вновь станет возможна ссылка на динамический массив, необходимо переопределить размерности переменной массива с помощью инструкции ReDim. Например:

```
Dim B() As Integer
ReDim B(6)
Erase B
ReDim B(3)
```

Константы

Константы, в отличие от переменных, не могут изменять свои значения. Использование констант делает программы легче читаемыми и проще исправляемыми, т. к. отпадает необходимость заменять многократно значения по тексту программы, а достаточно ввести новое значение при объявлении константы.

```
[Public | Private] Const constname [As type] = expression
```

- `Public` — ключевое слово, используемое на уровне модуля для объявления констант, доступных всем процедурам во всех модулях. Не допускается в процедурах;
- `Private` — ключевое слово, используемое на уровне модуля для объявления констант, доступных только внутри модуля, в котором выполняется описание. Не допускается в процедурах;
- *constname* — имя константы, удовлетворяющее стандартным правилам именования переменных;
- *type* — один из поддерживаемых типов данных: `Byte`, `Boolean`, `Integer`, `Long`, `Currency`, `Single`, `Double`, `Date`, `String` или `Variant`. Для каждой объявляемой константы следует использовать отдельное предложение `As` *type*;
- *expression* — литерал, другая константа или любое выражение, которое включает все арифметические или логические операторы за исключением `Is`.

В данном коде определяются числовая и строковая константы:

```
Private Const Rate As Single = 0.2
Public Const FirmName = "Транспортное агентство: Быстроногие зайцы"
```

Перечисляемый тип

Перечисляемый тип предоставляет удобный способ работы с константами и позволяет ассоциировать значения констант с их именами.

```
[Public | Private] Enum Name
   MemberName [= ConstantExpression]
   MemberName [= ConstantExpression]
...
End Enum
```

- *Name* — имя перечисляемого типа;
- *MemberName* — имя константы. По умолчанию значение первой константы равно 0, второй 1 и т. д. Через параметр *ConstantExpression* константам можно назначать произвольные значения;
- *ConstantExpression* — значение константы.

В следующем примере (листинг 4.3) задается перечисляемый тип для идентификации стороны монеты.

Листинг 4.3. Бросание монеты

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

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

Тип данных, определенный пользователем

Наряду с массивами, представляющими нумерованный набор элементов одного типа, существует еще один способ создания структурного типа — определенный пользователем тип, или, в привычной терминологии для программистов, — запись. Запись — это совокупность нескольких элементов, каждый из которых может иметь свой тип. Элемент записи называется полем. Запись является частным случаем класса, в котором не определены свойства и методы.

```
Private | Public] Type varname
   elementname [([subscripts])] As type
   elementname [([subscripts])] As type
...
End Type
```

- `Private` — используется для описания определяемых пользователем типов, которые доступны только в модуле, содержащем описание;
- `Public` — используется для описания определяемых пользователем типов, которые доступны для всех процедур во всех модулях всех проектов;
- *`varname`* — имя типа, определяемого пользователем;
- *`elementname`* — имя элемента, определяемого пользователем типа;
- *`subscripts`* — размерности элемента, являющегося массивом.
- *`type`* — тип данных элемента; поддерживаются типы `Byte`, `Boolean`, `Integer`, `Long`, `Currency`, `Single`, `Double`, `Date`, `String` (для строк переменной длины), `String * длина` (для строк фиксированной длины), `Object`, `Variant`, другой определяемый пользователем тип или объектный тип.

Инструкция `Type` используется только на уровне модуля. При появлении в модуле класса инструкции `Type` должно предшествовать ключевое слово `Private`.

В следующем примере (листинг 4.4) инструкция `Type` используется для определения типа данных, инкапсулирующего в себе информацию о сотруднике некоторой фирмы.

Листинг 4.4. Пример типа, определенного пользователем

```
Type Employee
   FirstName As String
   LastName As String
   Position As String
   BirthDate As Date
End Type

Sub InitialData()
   Dim emp As Employee
   With emp
      .FirstName = "James"
      .LastName = "Bond"
      .Position = "Secret agent 007"
      .BirthDate = #17/05/60#
   End With
   With emp
      MsgBox .FirstName & vbCr & _
             .LastName & vbCr & _
             .Position & vbCr & _
             .BirthDate
   End With
End Sub
```

Можно создавать массив, содержащий элементы собственного типа. Например, следующий массив (листинг 4.5) состоит из сведений о 20 сотрудниках отдела продаж некоторой фирмы.

Листинг 4.5. Пример массива, элементы которого имеют тип, определенный пользователем

```
Sub InitialData()
   Dim emp(3) As Employee
   With emp(0)
      .FirstName = "James"
      .LastName = "Bond"
      .Position = "Secret agent 007"
      .BirthDate = #17/05/60#
   End With
   With emp(1)
      .FirstName = "Alice"
      .LastName = "Smith"
      .Position = "Just a secret agent"
      .BirthDate = #06/09/69#
   End With
End Sub
```

Операции VBA

В программах на VBA можно использовать стандартный набор операций над данными. Имеются три типа операций:

- *математические* — выполняются над числами, и их результатом являются числа;
- *отношения* — применяются не только к числам, и их результатом являются логические значения, например `x>y`;
- *логические* — применяются логическими выражениями, и их результатом являются логические значения, например `Not x And y`.

Математические операции

В VBA поддерживается стандартный набор математических операций от сложения до возведения в степень (табл. 4.3).

Таблица 4.3. *Математические операции*

Операция	Описание
exp1 + *exp2*	Сложение
exp1 – *exp2*	Вычитание
-*exp*	Перемена знака
exp1 * *exp2*	Умножение
exp1 / *exp2*	Деление
exp1 \ *exp2*	Целочисленное деление
exp1 Mod *exp2*	Остаток от деления по модулю
exp1 ^ *exp2*	Возведение в степень

Операции отношения

Перечислим в табл. 4.4 операции отношения, используемые в VBA.

Таблица 4.4. *Операции отношения*

Операция	Описание
exp1 < *exp2*	Меньше
exp1 > *exp2*	Больше
exp1 <= *exp2*	Меньше или равно
exp1 >= *exp2*	Больше или равно
exp1 <> *exp2*	Не равно
exp1 = *exp2*	Равно
exp1 Is *exp2*	Сравнение двух операндов, содержащих ссылки на объекты
exp1 Like *exp2*	Сравнение двух строковых выражений

Логические операции

Перечислим в табл. 4.5 логические операции, используемые в VBA.

Таблица 4.5. *Логические операции*

Операция	Описание
exp1 And *exp2*	Логическое умножение
exp1 Or *exp2*	Логическое сложение

Таблица 4.5 (окончание)

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

Директива *Option Compare*

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

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

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

В следующем примере (листинг 4.6) в поле ввода диалогового окна пользователь должен ввести свое имя латинскими буквами. При нажатии кнопки **OK** программа анализирует информацию и отображает сообщение. А именно:

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

Листинг 4.6. Директива Option Compare

```
Option Compare Text

Sub DemoCompare()
   Dim name As String, lng As Integer, i As Integer
   name = InputBox("Введите имя")
   lng = Len(Trim(name))
```

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

Приоритеты операций

VBA выполняет операции в соответствии с их приоритетами, что обеспечивает однозначность в трактовании значений выражений. В табл. 4.6 перечислены приоритеты выполнения операций.

Таблица 4.6. *Приоритеты выполнения операций*

Приоритет	Операция
1	Вызов функции и скобки
2	^
3	– (смена знака)
4	* и /
5	\ (деление нацело)
6	Mod (остаток от деления нацело)
7	+ и –
8	>, <, >=, <=, <> и =
9	Not
10	And
11	Or
12	Xor
13	Equ
14	Imp

Встроенные функции VBA

В VBA имеется большой набор встроенных функций и процедур, использование которых существенно упрощает программирование. Эти функции можно разделить на следующие основные категории:

- ❒ математические функции;
- ❒ функции проверки типов;
- ❒ функции преобразования форматов;
- ❒ функции обработки строк;
- ❒ функции времени и даты;
- ❒ финансовые функции.

В последующих разделах рассмотрены основные функции из этих категорий.

Математические функции

В VBA имеется большой список математических функций от нахождения косинуса числа до генерации случайного числа, позволяющих произвести любые вычисления. В табл. 4.7 перечислены математические функции VBA.

Таблица 4.7. *Математические функции VBA*

Функция	Описание
`Abs`	Модуль (абсолютная величина) числа
`Atn`	Арктангенс
`Cos`	Косинус
`Exp`	Экспонента, т. е. результат возведения основания натурального логарифма в указанную степень
`Log`	Натуральный логарифм
`Rnd`	Возвращает очередное число из последовательности псевдослучайных чисел из интервала [0, 1]. `Rnd(`*`number`*`)` Если значение параметра *`number`* меньше нуля, то `Rnd` возвращает каждый раз одно и то же число. Если значение параметра *`number`* больше нуля или этот параметр опущен, то `Rnd` возвращает следующее случайное число в последовательности. Если значение параметра *`number`* равняется нулю, то `Rnd` возвращает случайное число, возвращенное при предыдущем вызове этой функции

Таблица 4.7 *(окончание)*

Функция	Описание
Sgn	Знак числа
Sin	Синус
Sqr	Квадратный корень из числа
Tan	Тангенс
Fix, Int	Обе функции Int и Fix отбрасывают дробную часть числа и возвращают целое значение. Различие между функциями Int и Fix состоит в том, что для отрицательного значения параметра число функция Int возвращает ближайшее отрицательное целое число, меньшее либо равное указанному, а Fix — ближайшее отрицательное целое число, большее либо равное указанному

Создание последовательности случайных чисел

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

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

Повторение последовательности случайных чисел

Если есть необходимость повторить ту же последовательность случайных чисел, то перед вызовом оператора Randomize с параметром, надо вызвать функцию Rnd с отрицательным значением параметра. Например, следующий код выводит дважды одну и ту же последовательность случайных чисел.

```
Dim i As Integer
Dim j As Integer
For i = 1 To 2
   Debug.Print Rnd(-1)
   Randomize 0.3
```

```
   For j = 1 To 3
     Debug.Print Rnd()
   Next
Next
```

Получение случайного числа из целочисленного интервала

Если требуется получить случайные числа не из интервала [0, 1], а из целочисленного интервала, например [1, 6], можно воспользоваться функцией `RndInt`, определенной в следующем коде (листинг 4.7).

Листинг 4.7. Получение случайного числа из целочисленного интервала

```
Sub DemoIntegerRnd ()
   Dim i As Integer
   For i = 0 To 10
      Debug.Print RndInt(1, 6)
   Next
End Sub

Function RndInt(ByVal lowerbound As Integer, _
                ByVal upperbound As Integer) As Integer
   Randomize
   RndInt = (upperbound – lowerbound) * Rnd() + lowerbound
End Function
```

Как найти значение числа π

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

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

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

Функции проверки типов

Функция проверки типа проверяет, является ли переменная выражением специфицированного типа. В табл. 4.8 приведены используемые в VBA функции проверки типов.

Таблица 4.8. *Функции проверки типов*

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

Преобразование строки в число

Функция `Val` возвращает подходящее число, содержащееся в строке, указанной в качестве значения ее параметра. В качестве допустимого десятичного разделителя функция `Val` воспринимает только точку. При наличии другого десятичного разделителя (например, запятой) для преобразования чисел в строки следует использовать функцию `CStr`, описанную далее. Следующий код (листинг 4.8) демонстрирует результаты преобразования строки в число.

Листинг 4.8. Преобразование строки в число

```
Dim s As String
s = "23432"
Debug.Print Val(s) ' Отобразится 23432
s = "123-45-45"
Debug.Print Val(s) ' Отобразится 123
s = #12/15/1999#
Debug.Print Val(s) ' Отобразится 15.12
s = "198,005"
Debug.Print Val(s) ' Отобразится 198
s = "198.005"
Debug.Print Val(s) ' Отобразится 198.005
```

Преобразование числа в строку

Функция `Str` возвращает значение типа `Variant` (`String`), являющееся строковым представлением числа. Функция `CStr` преобразует значение в строку, т. е. в значение типа `String`. Например, следующий код проверяет, входит

ли 0 в сгенерированное целое случайное число из интервала от 1 до 1000. Для этого число преобразуется в строку, а затем с помощью функции `InStr` проверяется, является ли указанная строка подстрокой данной.

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

Другие функции преобразования типов

Кроме функций `Val` и `Str`, в VBA имеется целый ряд функций преобразования типов данного выражения в указанный (табл. 4.9).

Таблица 4.9. *Функции преобразования типов*

Функция	Тип, в который преобразуется выражение
CBool	Boolean
CByte	Byte
CCur	Currency
CDate	Date
CDbl	Double
CDec	Decimel
CInt	Integer
CLng	Long
CSng	Single
CVar	Variant
CStr	String

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

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

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

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

Форматирование числа

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

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

- *`Expression`* — любое допустимое выражение;
- *`Format`* — любое допустимое именованное или определяемое пользователем выражение формата. Примером именованного формата является `Fixed` — формат действительного числа с двумя значащими цифрами после десятичной точки. Примеры именованных форматов даны в табл. 4.10 и 4.11;
- *`FirstDayOfWeek`* — константа, определяющая первый день недели;
- *`FirstWeekOfYear`* — константа, определяющая первую неделю года.

Таблица 4.10. *Именованные числовые форматы*

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

Таблица 4.10 *(окончание)*

Имя формата	Описание
`True/False`	Отображает `False`, если число равно 0, и `True` — в противном случае
`On/Off`	Отображает `Off`, если число равно 0, и `On` — в противном случае

Таблица 4.11. *Именованные форматы даты и времени*

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

Например, в результате действия приводимого ниже кода (листинг 4.9) в окно **Immediate** будут выведены следующие значения:

```
General Number          4654646.544564
Currency                4 654 646.54ð.
Fixed                   4654646.54
Standard                4 654 646.54
Percent                 465464654.46%
Scientific              4.65E+06
Yes/No                  Да
True/False              Истина
On/Off                  Вкл
General Date            12.08.2002 16:15:41
Long Date               12 Август 2002 г.
```

```
Medium Date              12-авг-02
Short Date               12.08.2002
Long Time                16:15:41
Medium Time              04:15
Short Time               16:15
```

Листинг 4.9. Примеры именованных форматов

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

Пользовательские форматы

Пользовательские форматы позволяют настроить вид отображаемых значений так, как желает пользователь. При построении пользовательского формата применяются специальные символы, перечисленные в табл. 4.12.

Таблица 4.12. *Символы, используемые при построении пользовательского формата*

Символ	Описание
0	Резервирует позицию цифрового разряда. Отображает цифру или нуль. Если у числа, представленного параметром, есть какая-нибудь цифра в той позиции разряда, в которой в строке формата находится "0", функция отображает эту цифру параметра, если нет — в этой позиции отображается нуль

Таблица 4.12 *(окончание)*

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

В табл. 4.13 приведен ряд примеров использования пользовательских форматов.

Таблица 4.13. *Примеры пользовательских форматов*

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

Форматирование чисел

Для форматирования чисел в VBA имеется специализированная функция FormatNumber.

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

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

В табл. 4.14 приведен ряд примеров использования функции FormatNumber.

Таблица 4.14. *Примеры использования функции* *FormatNumber*

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

Форматирование процентов

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

Например, следующие две инструкции

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

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

Денежный формат

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

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

Форматирование даты и времени

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

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

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

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

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

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

Функции обработки строк

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

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

Все эти функции с примерами подробно опишем в следующих разделах.

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

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

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

Функция `Chr` преобразует ASCII-код в строку.

```
Chr(Charcode)
```

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

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

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

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

Преобразование строки к нижнему или верхнему регистру

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

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

Возвращение подстроки по указанному местоположению

Функции `Left` и `Right` возвращают подстроки, состоящие из заданного числа соответственно первых и последних символов данной строки.

```
Left(String, Length)
Right(String, Length)
```

- *String* — исходная строка;
- *Length* — число символов.

Функция Mid возвращает подстроку строки, содержащую указанное число символов.

```
Mid(String, Start[, Length])
```

- *String* — строка, из которой извлекается подстрока;
- *Start* — позиция символа в строке *String*, с которого начинается нужная подстрока;
- *Length* — число возвращаемых символов подстроки.

Например:

```
Debug.Print Left("Hello, World", 5) ' Отобразится Hello
Debug.Print Right("Hello, World", 5) ' Отобразится World
Debug.Print Mid("Hello, World", 8, 2) ' Отобразится Wo
```

Определение длины строки

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

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

Копия строки без начальных и конечных пробелов

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

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

Строка, состоящая из указанного числа пробелов

Функция Space возвращает строку, состоящую из указанного числа пробелов. Например, в следующем коде (листинг 4.10) за счет последовательного

добавления перед данной строкой другой, состоящей каждый раз из меньшего числа пробелов, создается эффект бегущей строки.

Листинг 4.10. Бегущая строка

```
Private n As Integer

Sub LetsGo()
   n = 10
   RunString
End Sub

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

Строка, состоящая из указанного числа повторяющихся символов

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

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

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

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

Преобразование строки в массив

Функция `Split` преобразует строку в массив.

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

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

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

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

Преобразование массива в строку

Функция `Join` преобразует массив в строку.

```
Join(SourceArray, [Delimiter])
```

- *SourceArray* — преобразуемый массив;
- *Delimiter* — разделитель между элементами строкового выражения, из которых создается массив. По умолчанию используется пробел.

Например:

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

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

Сравнение строк

Функция `StrComp` возвращает результат сравнения двух строк.

```
StrComp(String1, String2[, Compare])
```

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

Функция `StrComp` возвращает следующие значения:

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

Например:

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

Создание зеркальной строки по отношению к данной

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

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

Нахождение вхождения в строку подстроки

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

```
InStr([Start, ] StringCheck, StringMatch[, Compare])
InStrRev(StringCheck, StringMatch, [Start], [Compare])
```

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

Например:

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

Замена в строке подстроки

Функция `Replace` заменяет в строке подстроку.

```
Replace(Expression, Find, Replacewith[, Start[, Count [, Compare]]])
```

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

Например:

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

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

Функции, возвращающие строки

Некоторые функции имеют по две версии, одна из которых возвращает тип данных Variant, а другая — тип данных String. Первая версия является более удобной, т. к. при этом для значений типа Variant преобразование типов данных выполняется автоматически. Вторая версия, возвращающая тип String, использует меньше памяти и может быть полезна для ее экономии, если в программе имеется очень много переменных.

Следующие функции возвращают значения типа String, если к имени функции добавляется символ доллара ($). Эти функции имеют такое же применение и синтаксис, как и их эквиваленты без символа доллара, возвращающие тип Variant:

Chr$	Error$	LCase$	Right$	String$
CurDir$	Format$	Left$	RTrim$	Time$
Date$	Input$	LTrim$	Space$	Trim$
Dir$	InputB$	Mid$	Str$	UCase$

Функции времени и даты

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

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

В последующих разделах на примерах рассмотрены все эти функции.

Определение текущей системной даты

Функция `Date` возвращает значение типа `Variant` (`Date`), содержащее текущую системную дату. Например, данная строка писалась 12 августа 2002 г., поэтому инструкция

```
Debug.Print Date
```

выводит значение `12.08.2002`.

Определение текущего системного времени

Функция `Time` возвращает значение типа `Variant` (`Date`), содержащее текущее время по системным часам компьютера. Например, данная строка писалась в 20 часов 46 минут и 14 секунд 12 августа 2002 г., поэтому инструкция

```
Debug.Print Time
```

выводит значение `20:46:14`.

Определение текущей даты и системного времени

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

```
Debug.Print Now
```

выводит значение `12.08.2002 20:49:24`.

Извлечение из времени часового, минутного и секундного компонентов

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

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

Извлечение из даты годового, месячного и дневного компонентов

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

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

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

Определение дня недели

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

```
Weekday(Date, [Firstdayofweek])
```

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

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

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

Определение числа секунд, прошедших с полуночи

Функция `Timer` возвращает значение типа `Single`, представляющее число секунд, прошедших после полуночи. В Windows, в отличие от Macintosh, функция `Timer` возвращает даже не число секунд, а число долей секунд, прошедших после полуночи. Например, следующий код (листинг 4.11) создает мигающую ячейку, у которой цвет фона в течение 5 секунд попеременно становится то красным, то зеленым. Для того чтобы во время выполнения цикла компьютер не зависал, и приложение могло реагировать на различные события, происходящие в системе, в циклы добавлены операторы `DoEvents`.

Листинг 4.11. Мигающая ячейка

```
Sub LetsGo()
   Dim fl As Boolean
   Dim old As Long
   old = Timer
   Do
      fl = Not fl
      If fl Then
         Range("A1").Interior.Color = RGB(255, 0, 0)
      Else
         Range("A1").Interior.Color = RGB(0, 255, 0)
      End If
      Delay
      DoEvents
   Loop While Timer - old <= 5
   Range("A1").Interior.ColorIndex = xlNone
End Sub

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

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

Функция `DateDiff` возвращает значение типа `Variant` (`Long`), указывающее количество временных интервалов, прошедших между двумя датами.

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

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

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

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

Определение компонента даты

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

```
DatePart(Interval, Date[, Firstdayofweek[, Firstweekofyear]])
```

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

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

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

Добавление к дате указанного временного интервала

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

```
DateAdd(Interval, Number, Date)
```

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

В данном примере переменной d присваивается значение 17.09.1968.

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

Преобразование часов, минут и секунд в формат времени

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

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

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

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

Преобразование года, месяца и дня в формат даты

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

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

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

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

Преобразование строки в формат времени

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

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

Примечание

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

Функции выбора

Кроме перечисленных выше основных категорий функций в VBA имеются также функции выбора, которые дублируют некоторые аспекты операторов условного перехода. В табл. 4.15 приведены эти функции.

Таблица 4.15. *Функции выбора*

Функция	Описание

`IIf`	Возвращает одну из двух альтернатив.
	`IIf(`*`Expr`*`, `*`Truepart`*`, `*`Falsepart`*`)`
	♦ *`Expr`* — проверяемое выражение;
	♦ *`Truepart`* — значение или выражение, возвращаемое, если параметр *`Expr`* принимает значение `True`;
	♦ *`Falsepart`* — значение или выражение, возвращаемое, если параметр *`Expr`* принимает значение `False`.
	В следующем примере выводится сообщение, меньше или больше, чем 0,5, сгенерированное случайное число:
	`Debug.Print IIf(Rnd() < 0.5, "< 0.5", ">=0.5")`

`Choose`	Возвращает значение, выбранное из списка параметров.
	`Choose(`*`Index`*`, `*`Choice1`*`[, `*`Choice2`*`, ... [, `*`ChoiceN`*`]])`
	♦ *`Index`* — числовое выражение или поле, значением которого является число, лежащее между 1 и числом элементов в списке;
	♦ *`Choice`* — выражение типа `Variant`, содержащее один из элементов списка.
	Если значение параметра *`Index`* равно 1, то функция возвращает первый элемент списка, если равняется 2, возвращается второй элемент и т. д.
	Например, в следующем коде в зависимости от сгенерированного случайного целого числа из интервала [1, 3] будет выводиться `"один"`, `"два"` или `"три"`.
	`Dim r As Integer`
	`r = 2 * Rnd() + 1`
	`Debug.Print Choose(r, "один", "два", "три")`

Таблица 4.15 *(окончание)*

Функция	Описание

`Switch`	Возвращается значение, соответствующее первому истинному выражению в списке.
	`Switch(`*`Expr1`*`, `*`Value1`*`[, `*`Expr2`*`, `*`Value2`*` ... [, `*`ExprN`*`, `*`ValueN`*`]])`
	♦ *`Expr`* — выражение типа `Variant`, подлежащее вычислению;
	♦ *`Value`* — возвращаемое значение или выражение, если соответствующее *`Expr`* принимает значение `True`.
	Например, в следующем коде отобразится `Agent 007`.
	`Sub DemoSwitch()`
	`  Debug.Print WhoIsWho("Bond")`
	`End Sub`
	`Function WhoIsWho(LastName As String)`
	`  WhoIsWho = Switch(LastName = "Bond", "Agent 007", _`
	`  LastName = "Milne", "Writer")`
	`End Function`

Встроенные диалоговые окна

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

Окно ввода

Функция `InputBox` выводит на экран диалоговое окно, содержащее сообщение, поле ввода и две кнопки **OK** и **Cancel**. Устанавливает режим ожидания ввода текста пользователем и нажатия кнопки, а затем, по нажатии на кнопку **OK**, возвращает значение типа `String`, содержащее текст, введенный в поле ввода. При нажатии кнопки **Cancel** возвращает пустую строку (`Empty`).

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

♦ *`Prompt`* — строковое выражение, отображаемое как сообщение в диалоговом окне. Строковое выражение *`Prompt`* может содержать несколько строк. Для разделения строк допускается использование символа возвра-

та каретки (`Chr(13)`), символа перевода строки (`Chr(10)`) или комбинации этих символов (`Chr(13) & Chr(10)`);

- *`Title`* — строковое выражение, отображаемое в строке заголовка диалогового окна. Если этот параметр опущен, то в строку заголовка помещается имя приложения;
- *`Default`* — строковое выражение, отображаемое в поле ввода как используемое по умолчанию, если пользователь не введет другую строку. Если этот параметр опущен, то поле ввода изображается пустым;
- *`Xpos`* — числовое выражение, задающее расстояние по горизонтали между левой границей диалогового окна и левым краем экрана. Если этот параметр опущен, то диалоговое окно выравнивается по центру экрана по горизонтали;
- *`Ypos`* — числовое выражение, задающее расстояние по вертикали между верхней границей диалогового окна и верхним краем экрана. Если этот параметр опущен, то диалоговое окно помещается по вертикали примерно на одну треть высоты экрана;
- *`Helpfile`* — строковое выражение, определяющее имя файла справки, содержащего справочные сведения о данном диалоговом окне. Если этот параметр указан, то необходимо указать также параметр `Context`;
- *`Context`* — числовое выражение, определяющее номер соответствующего раздела справочной системы. Если этот параметр указан, то необходимо указать также параметр *`Helpfile`*.

Например, следующий код отображает на экране окно ввода, показанное на рис. 4.1.

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

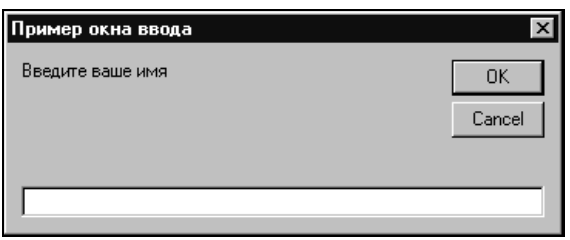


Рис. 4.1. Окно ввода

Как обработать нажатие кнопки *Cancel*

При вводе данных при помощи функции `InputBox` разумно в программе предусмотреть обработку события нажатия кнопки **Cancel** окна ввода. Например, в следующей простой ситуации

```
x = InputBox("Введите x", "Пример")
y = x ^ 2
```

в том случае, если пользователь нажмет кнопку **Cancel**, произойдет прерывание выполнения кода с сообщением об ошибке несовпадения типов. Этой ситуации легко избежать, добавив только одну строку кода, проверяющую, не введена ли в поле ввода окна ввода пустая строка (что, собственно говоря, и происходит при нажатии кнопки **Cancel**).

```
Sub DemoInputBox2()
   Dim x As String
   Dim y As Double
   x = InputBox("Введите x", "Пример")
   If x = Empty Then Exit Sub
   y = x ^ 2
   Debug.Print y
End Sub
```

Окно сообщения

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

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

- *Prompt* — строковое выражение, отображаемое как сообщение в диалоговом окне;
- *Buttons* — числовое выражение, представляющее сумму значений, которые указывают число и тип отображаемых кнопок, тип используемого значка, основную кнопку и модальность окна сообщения. Значение по умолчанию этого параметра равняется 0. Значения констант, определяющих число, тип кнопок и тип используемого значка приведены в табл. 4.16—4.18;
- *Title* — строковое выражение, отображаемое в строке заголовка диалогового окна. Если этот параметр опущен, то в строку заголовка помещается имя приложения;
- *Helpfile* — строковое выражение, определяющее имя файла справки, содержащего справочные сведения о данном диалоговом окне. Если этот параметр указан, необходимо указать также параметр *Context*;

- `Context` — числовое выражение, определяющее номер соответствующего раздела справочной системы. Если этот параметр указан, то необходимо указать также параметр `Helpfile`.

Таблица 4.16. *Значения параметра* `Buttons` *процедуры* `MsgBox`*, определяющие кнопки, отображаемые в диалоговом окне*

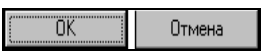
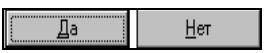
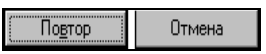
Константа	Значение	Отображаются кнопки
vbOKOnly	0	
vbOKCancel	1	
vbAbortRetryIgnore	2	
vbYesNoCancel	3	
vbYesNo	4	
vbRetryCancel	5	

Таблица 4.17. *Значения параметра* `Buttons` *процедуры* `MsgBox`*, определяющие отображаемые в диалоговом окне информационные значки*

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

Таблица 4.18. *Значения параметра* `Buttons` *процедуры* `MsgBox`*, определяющие основную кнопку в диалоговом окне*

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

При написании программ с откликом в зависимости от того, какая кнопка диалогового окна нажата, вместо возвращаемых значений удобнее использовать константы VBA, перечисленные в табл. 4.19, которые делают код программы более прозрачным для чтения и, к тому же, их легко запомнить.

Таблица 4.19. *Константы, идентифицирующие нажатую кнопку*

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

Приведем демонстрационный пример использования окон сообщений. В этом примере (листинг 4.12) вычисляется квадрат введенного значения, и найденное значение затем последовательно отображается в пяти окнах сообщения. Первое окно является простым окном с сообщением (рис. 4.2), второе — с сообщением, выводимым в две строчки (рис. 4.3), третье — с сообщением и информационным значком (рис. 4.4), четвертое — с сообщением, информационным значком и двумя кнопками, причем кнопка **Yes** установлена как кнопка, выполняемая по умолчанию (рис. 4.5), пятое — с сообщением, информационным значком и пользовательским заголовком (рис. 4.6).

Листинг 4.12. Пример работы со встроенными окнами

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



Рис. 4.2. Простое окно сообщения

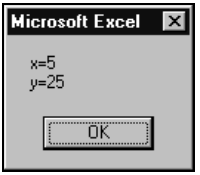


Рис. 4.3. Окно с сообщением, выводимым в две строчки



Рис. 4.4. Окно с сообщением и информационным значком

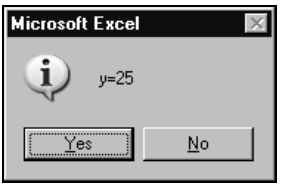


Рис. 4.5. Окно с сообщением, информационным значком и двумя кнопками

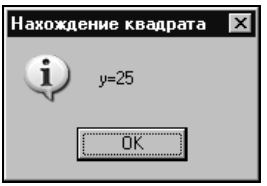


Рис. 4.6. Окно с сообщением, информационным значком и пользовательским заголовком

Определение нажатой кнопки в окне ввода

Процедура `MsgBox` удобна для вывода той или иной информации. Однако если необходимо узнать, какой выбор сделал пользователь при нажатии отображаемых в диалоговом окне кнопок, то процедуру `MsgBox` нужно использовать как функцию. В этом случае значение, возвращаемое `MsgBox`, надо присваивать какой-то переменной, а ее параметры заключать в скобки. Рассмотрим пример использования функции `MsgBox` с анализом нажатой кнопки (листинг 4.13). В этом примере на экране отображается диалоговое окно с тремя кнопками **Yes**, **No** и **Cancel** и информационным значком. Клавише <Enter> назначена функция кнопки **Yes**. По нажатии одной из этих кнопок на экране отображается сообщение, подтверждающее нажатие.

Листинг 4.13. Пример использования окна сообщений с тремя кнопками

```
Sub DemoThreeButtonsMsgBox()
   Dim structure As Integer
   Dim btn As Integer
   structure = vbYesNoCancel + vbQuestion + vbDefaultButton1
   btn = MsgBox("Выберите <Yes>, <No> или <Cancel>", structure, _
               "Еще пример")
   Select Case btn
     Case vbYes
        MsgBox "Выбрали <Yes>", vbInformation, "Еще пример"
```

```
    Case vbNo
      MsgBox "Выбрали <No>", vbInformation, "Еще пример"
     Case vbCancel
      MsgBox "Выбрали <Cancel>", vbInformation, "Еще пример"
  End Select
End Sub
```

Финансовые функции

В VBA имеется довольно много встроенных функций для проведения финансовых расчетов. Они существенно облегчают разработку финансовых приложений. В табл. 4.20 приведен список этих функций.

Таблица 4.20. *Финансовые функции*

Функция	Описание
DDB	Возвращает величину амортизации имущества для указанного периода, используя метод двукратного (или *k*-кратного) учета амортизации. DDB(*cost*, *salvage*, *life*, *period*[, *factor*]) ♦ *cost* — начальная стоимость имущества; ♦ *salvage* — остаточная стоимость в конце амортизации (иногда называется ликвидной стоимостью имущества); ♦ *life* — количество периодов, за которые собственность амортизируется (иногда называется периодом амортизации); ♦ *period* — номер периода для вычисления амортизации (должен быть измерен в тех же единицах, что и время полной амортизации); ♦ *factor* — норма снижения балансовой стоимости (амортизации). Если *factor* опущен, то предполагается, что он равен 2 (метод двукратного учета амортизации)
FV	Возвращает будущее значение вклада на основе периодических постоянных платежей и постоянной процентной ставки. Функция FV подходит для расчета итогов накоплений при ежемесячных банковских взносах. FV(*rate*, *nper*, *pmt*[, *pv*[, *type*]]) ♦ *rate* — процентная ставка за период; ♦ *nper* — общее число периодов выплат; ♦ *pmt* — величина постоянных периодических платежей; ♦ *pv* — текущее значение, т. е. общая сумма, которую составят будущие платежи; ♦ *type* — если значение параметра *type* равно 0 или он опущен, то оплата в конце периода, если 1, то в начале периода

Таблица 4.20 *(продолжение)*

Функция	Описание
IPmt	Возвращает платежи по процентам за данный период на основе периодических постоянных выплат и постоянной процентной ставки. IPmt(*rate*, *per*, *nper*, *pv*[, *fv*[, *type*]]) ♦ *rate* — процентная ставка за период; ♦ *per* — период, для которого требуется найти прибыль; должен находиться в интервале от 1 до *nper*; ♦ *nper* — общее число периодов выплат; ♦ *pv* — текущее значение, т. е. общая сумма, которую составят будущие платежи; ♦ *fv* — будущая стоимость или баланс наличности, которого нужно достичь после последней выплаты. Если этот параметр опущен, то его значение полагается равным 0 (будущая стоимость займа, например, равна 0); ♦ *type* — если значение параметра *type* равно 0 или он опущен, то оплата в конце периода, если 1, то в начале периода
IRR	Возвращает внутреннюю скорость оборота для ряда последовательных операций с наличными, представленными числовыми значениями. Объемы операций не обязаны быть регулярными, как в случае ренты. Внутренняя скорость оборота — это процентная ставка дохода, полученного от инвестиций, состоящих из выплат (отрицательные значения) и поступлений (положительные значения), которые происходят в регулярные периоды времени. IRR(*values*()[, *guess*]) ♦ *values*() — массив, содержащий числовые величины, для которых вычисляется внутренняя скорость оборота средств. В этот массив должны входить по крайней мере одно положительное и одно отрицательное значение для того, чтобы можно было вычислить внутреннюю скорость оборота. IRR использует порядок *values* для интерпретации порядка денежных выплат или поступлений, поэтому необходимо вводить *values* выплат и поступлений в правильном порядке; ♦ *guess* — величина, о которой предполагается, что она близка к результату IRR
MIRR	Возвращает модифицированную внутреннюю скорость оборота средств для ряда последовательных периодических операций с наличными. Функция MIRR учитывает как стоимость инвестиции, так и доход, получаемый от реинвестирования. MIRR(*values*(), *finance_rate*, *reinvest_rate*) ♦ *values*() — массив, содержащий числовые величины. Эти числа представляют ряд денежных выплат (отрицательные значения) и поступлений (положительные значения), происходящие в регулярные периоды времени;

Таблица 4.20 *(продолжение)*

Функция	Описание
(прод.)	♦ *finance_rate* — норма прибыли, выплачиваемой за деньги, находящиеся в наличном обороте; ♦ *reinvest_rate* — норма прибыли, получаемой за деньги, находящиеся в наличном обороте при реинвестировании
NPer	Вычисляет общее количество периодов выплаты для данного вклада на основе периодических постоянных выплат и постоянной процентной ставки. NPer(*rate*, *pmt*, *pv*[, *fv*[, *type*]]) ♦ *rate* — процентная ставка за период; ♦ *pmt* — величина постоянных периодических платежей; ♦ *pv* — текущее значение, т. е. общая сумма, которую составят будущие платежи; ♦ *fv* — будущая стоимость или баланс наличности, которого нужно достичь после последней выплаты. Если *fv* опущено, оно полагается равным 0 (будущая стоимость займа, например, равна 0); ♦ *type* — если значение параметра *type* равно 0 или он опущен, то оплата в конце периода, если 1, то в начале периода
NPV	Возвращает чистый текущий объем вклада, вычисляемый на основе ряда последовательных поступлений наличных и нормы амортизации. NPV(*rate*, *values*()) ♦ *rate* — процентная ставка за период; ♦ *values*() — массив расходов и доходов, которые должны быть равномерно распределены по времени и осуществляться в конце каждого периода. Порядок элементов массива *values* определяет порядок поступлений и платежей
Pmt	Вычисляет величину постоянной периодической выплаты ренты (например, регулярных платежей по займу) при постоянной процентной ставке. Pmt(*rate*, *nper*, *pv*[, *fv*[, *type*]]) ♦ *rate* — процентная ставка за период; ♦ *nper* — общее число периодов выплат; ♦ *pv* — текущее значение, т. е. общая сумма, которую составят будущие платежи; ♦ *fv* — будущая стоимость или баланс наличности, которого нужно достичь после последней выплаты. Если *fv* опущено, оно полагается равным 0 (будущая стоимость займа, например, равна 0); ♦ *type* — если значение параметра *type* равно 0 или он опущен, то оплата в конце периода, если 1, то в начале периода

Таблица 4.20 *(продолжение)*

Функция	Описание
PPmt	Возвращает величину выплаты на данный период на основе периодических постоянных платежей и постоянной процентной ставки. PPmt(*rate*, *per*, *nper*, *pv*[, *fv*[, *type*]]) ♦ *rate* — процентная ставка за период; ♦ *per* — период, для которого требуется найти прибыль; должен находиться в интервале от 1 до *nper*; ♦ *nper* — общее число периодов выплат; ♦ *pv* — текущее значение, т. е. общая сумма, которую составят будущие платежи; ♦ *fv* — будущая стоимость или баланс наличности, которого нужно достичь после последней выплаты. Если *fv* опущено, оно полагается равным 0 (будущая стоимость займа, например, равна 0); ♦ *type* — если значение параметра *type* равно 0 или он опущен, то оплата в конце периода, если 1, то в начале периода
PV	Возвращает текущий объем вклада на основе постоянных периодических платежей. Функция PV аналогична NPV. Основное различие между функциями PV и NPV заключается в том, что PV допускает, чтобы денежные взносы происходили либо в конце, либо в начале периода. Кроме того, в отличие от денежных взносов переменной величины в функции NPV, денежные взносы в функции PV должны быть постоянны на весь период инвестиции. PV(*rate*, *nper*, *pmt*[, *fv*[, *type*]]) ♦ *rate* — процентная ставка за период; ♦ *nper* — общее число периодов выплат; ♦ *pmt* — величина постоянных периодических платежей; ♦ *fv* — будущая стоимость или баланс наличности, которого нужно достичь после последней выплаты. Если *fv* опущено, оно полагается равным 0 (будущая стоимость займа, например, равна 0); ♦ *type* — если значение параметра *type* равно 0 или он опущен, то оплата в конце периода, если 1, то в начале периода
Rate	Вычисляет процентную ставку за один период, необходимую для получения определенной суммы за заданный срок путем постоянных взносов. Функция Rate вычисляет процентную ставку методом итераций и может не найти решение. Если после 20 итераций погрешность определения ставки превышает 0.0000001, то функция Rate возвращает значение ошибки. Rate(*nper*, *pmt*, *pv*[, *fv*[, *type*[, *guess*]]]) ♦ *nper* — общее число периодов выплат; ♦ *pmt* — величина постоянных периодических платежей;

Таблица 4.20 *(окончание)*

Функция	Описание
(прод.)	♦ *pv* — текущее значение, т. е. общая сумма, которую составят будущие платежи; ♦ *fv* — будущая стоимость или баланс наличности, которого нужно достичь после последней выплаты. Если *fv* опущено, оно полагается равным 0 (будущая стоимость займа, например, равна 0); ♦ *type* — если значение параметра *type* равно 0 или он опущен, то оплата в конце периода, если 1, то в начале периода; ♦ *guess* — предполагаемая величина процентной ставки. Если этот параметр опущен, то данное значение полагается равным 10%. Если функция Rate не сходится, следует попытаться использовать различные значения *guess*. Обычно функция Rate сходится, если *guess* имеет значение между 0 и 1
SLN	Возвращает величину амортизации имущества за один период времени, используя метод равномерной амортизации. SLN(*cost*, *salvage*, *life*) ♦ *cost* — начальная стоимость имущества; ♦ *salvage* — остаточная стоимость в конце амортизации (иногда называется ликвидной стоимостью имущества); ♦ *life* — количество периодов, за которые собственность амортизируется (иногда называется периодом амортизации)
SYD	Возвращает амортизацию имущества для указанного периода. SYD(*cost*, *salvage*, *life*, *period*) ♦ *cost* — начальная стоимость имущества; ♦ *salvage* — остаточная стоимость в конце амортизации (иногда называется ликвидной стоимостью имущества); ♦ *life* — количество периодов, за которые собственность амортизируется (иногда называется периодом амортизации); ♦ *period* — номер периода для вычисления амортизации (должен быть измерен в тех же единицах, что и время полной амортизации)

Оператор присваивания

Оператор присваивания присваивает значение выражения переменной, константе или свойству объекта. Оператор присваивания всегда включает знак равенства (=).

```
varname = expression
```

Оператор присваивания предписывает выполнить выражение *expression*, заданное в его правой части, и присвоить результат переменной *varname*, имя которой указано в левой части. В результате, например, действия следующей пары операторов

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

переменной `x` будет присвоено 4.

Присваивание переменной ссылки на объект

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

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

В общем случае инструкция `Set` имеет следующий синтаксис.

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

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

Оператор *With*

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

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

Допустимо также использование вложенных операторов `With`:

```
With Range("A1")
    .Value = 3
```

```
    With .Font
        .Italic = True
        .Size = 12
        .Bold = True
        .Color = RGB(255, 30, 255)
    End With
End With
```

Операторы управления

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

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

Оператор условного перехода

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

```
If condition Then [statements] [Else elsestatements]
```

Если *condition* возвращает значение `True`, то выполняется *statements*, если `False`, то *elsestatements*. Ветвь `Else` является необязательной.

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

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

или, что равносильно

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

Допускается также использование блочной формы синтаксиса, которая часто упрощает восприятие оператора условного перехода.

```
If condition Then
    [statements]
[Else
   [elsestatements]
End If]
```

Приводимый выше пример со скидкой можно записать в следующей эквивалентной блочной структуре:

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

```
Else
   Скидка = 0
End If
```

или

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

Внимание!

Следующая запись не допускается синтаксисом VBA и вызовет генерацию ошибки.

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

Дерево условий может оказаться гораздо более сложным, чем просто проверка одного условия. В этом случае используется оператор `If Then ElseIf`, который позволяет проверять множественные условия. Следующий пример показывает порядок проверки условий.

```
If condition Then
   [statements]
[ElseIf condition-1 Then
   [elseifstatements-1]
...
[ElseIf condition-n Then
   [elseifstatements-n]
[Else
   [elsestatements]]
End If
```

Первым проверяется *condition*. Если оно возвращает значение `True`, то выполняются *statements* и далее выполнение программы продолжится со строки, которая следует за ключевым словом `End If`. Если *condition* возвращает значение `False`, то проверяется *condition-1* и т. д. пока не встретится условие, имеющее значение `True`, или пока не встретится ключевое слово `Else`.

В следующем примере (листинг 4.14) в зависимости от величины вводимого числа отображается сообщение о принадлежности числа:

- либо интервалу [0, 1];
- либо интервалу (1, 2];
- либо о не принадлежности числа этим двум интервалам.

Листинг 4.14. Определение, какому интервалу принадлежит вводимое число

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

Оператор выбора

Если нужно проверить несколько условий, то оператор `If — Then` может стать слишком сложным. В этом случае применяется оператор выбора. Оператор выбора выполняет одну из нескольких групп инструкций в зависимости от значения выражения.

```
Select Case testexpression
   [Case expressionlist-1
      [statements-1]]
   ...
   [Case expressionlist-n
      [statements-n]]
   [Case Else
      [elsestatements]]
End Select
```

- *testexpression* — это список с разделителями, состоящий из одной или нескольких форм следующего вида: *expression*, *expression* `To` *expression*, `Is` *comparisonoperator expression*, где *expression* — это любое числовое или строковое выражение. Ключевое слово `To` задает диапазон значений. При использовании ключевого слова `To` перед ним должно находиться меньшее значение. Ключевое слово `Is` с операторами сравнения *comparisonoperator* задает диапазон значений. Если ключевое слово `Is` не указано, оно вставляется по умолчанию;
- *statements* — это одна или несколько инструкций, выполняемых в том случае, если выражение совпадает с любым компонентом списка от *expressionlist-1* до *expressionlist-n*;
- *elsestatements* — одна или несколько инструкций, выполняемых в том случае, если выражение не совпадает ни с одним из предложений `Case`.

Оператор выбора очень эффективен, когда надо проверить одну переменную, принимающую несколько значений. В следующем примере (листинг 4.15) в зависимости от величины введенного числа отображается сообщение, указывающее на величину числа или диапазон, которому оно принадлежит.

Листинг 4.15. Пример использования оператора выбора `Select Case`

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

В следующем примере (листинг 4.16) оператор `Select Case` в процедуре обработки события `KeyDown` пользовательской формы используется для идентификации нажатой клавиши.

Листинг 4.16. Идентификация нажатой клавиши при помощи оператора `Select Case`

```
Private Sub UserForm_KeyDown(ByVal KeyCode As MSForms.ReturnInteger, _
                             ByVal Shift As Integer)
   Select Case KeyCode
   Case vbKey0
      MsgBox "Нажали 0"
   Case vbKey1, vbKey2
      MsgBox "Нажали 1 или 2"
   Case vbKey3 To vbKey9, vbKeyA To vbKeyZ
      MsgBox "Нажали от 3 до 9 или буквенную клавишу"
   Case Else
      MsgBox "Нажата не буква, не цифра"
   End Select
End Sub
```

Оператор *For Next*

Оператор `For Next` повторяет выполнение группы инструкций указанное число раз.

```
For counter = start To end [Step step]
   [statements]
   [Exit For]
   [statements]
Next [counter]
```

Оператор `For Next` повторяет выполнение группы инструкций, пока переменная цикла *counter* изменяется от начального значения *start* до конечного значения *end* с указанным шагом *step*. Если шаг не указан, то он полагается равным 1. Альтернативный способ выхода из цикла предоставляет инструкция `Exit For`. В следующем примере (листинг 4.17) при помощи оператора цикла находится сумма элементов массива.

Листинг 4.17. Суммирование элементов массива

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

В следующем примере (листинг 4.18) находится произведение первых *n* натуральных чисел (*n*-факториал).

Листинг 4.18. Нахождение произведения первых *n* натуральных чисел

```
Sub Factor()
   n = 20
   Fact = 1
   For i = 1 To n
      Fact = Fact * i
   Next i
   MsgBox Format(Fact, "#############")
' Будет выведено 2432902008176640000
End Sub
```

В следующем примере (листинг 4.19) находится сумма значений из выделенного диапазона. При этом используются вложенные операторы циклов. Результат выводится в ячейки строки, находящейся непосредственно под выделенным диапазоном. А именно в ячейку под первым столбцом выделенного диапазона вводится строка "Сумма", а в соседнюю с ней справа ячейку выводится найденное значение.

Листинг 4.19. Суммирование элементов выделенного диапазона

```
Sub DemoSumSelection()
   Dim n As Integer, m As Integer, i As Integer, j As Integer
   Dim s As Double
   With Selection
      n = .Rows.Count
      m = .Columns.Count
   End With
   s = 0
   For i = 1 To n
      For j = 1 To m
        s = s + Selection.Cells(i, j).Value
      Next
   Next
   With Selection.End(xlDown)
      .Offset(1, 0).Value = "Сумма"
      .Offset(1, 1).Value = s
   End With
End Sub
```

Оператор *For Each*

Оператор `For Each` повторяет выполнение группы инструкций *statements* для каждого элемента *element* массива или семейства *group*.

```
For Each element In group
   [statements]
   [Exit For]
   [statements]
Next [element]
```

Альтернативный способ выхода из цикла предоставляет инструкция `Exit For`.

Использование оператора *For Each* при суммировании элементов массива

Оператор `For Each` можно использовать для суммирования элементов массива. Как это делается, показано в следующем коде (листинг 4.20).

Листинг 4.20. Суммирование элементов массива с помощью оператора `For Each`

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

Использование оператора *For Each* при переборе ячеек диапазона

Оператор `For Each` можно использовать при переборе ячеек диапазона и, в частности, при нахождении суммы значений, введенных в ячейки диапазона. Как это делается, продемонстрировано в следующем коде (листинг 4.21), в котором производится суммирование значений из выделенного диапазона.

Листинг 4.21. Суммирование значений из диапазона ячеек с помощью оператора `For Each`

```
Sub DemoSumSelection()
   Dim c As Range
   Dim s As Double
   s = 0
   For Each c In Selection.Cells
      s = s + c.Value
   Next
   MsgBox s
End Sub
```

В следующем примере (листинг 4.22) цикл `For Each` используется для изменения цвета ячеек, а именно все ячейки диапазона **A1:C4** с положительными значениями окрашиваются в синий цвет, а с неположительными — в красный.

Листинг 4.22. Изменение цвета ячеек

```
Sub DemoChangeColor()
   Dim c As Range
   For Each c In Range("A1:C4").Cells
      With c
         If .Value <= 0 Then
            .Interior.ColorIndex = 3
         Else
            .Interior.ColorIndex = 5
         End If
      End With
   Next
End Sub
```

Использование оператора *For Each* при переборе элементов семейства

Оператор `For Each` можно использовать при переборе элементов семейства. Например, следующий код (листинг 4.23) демонстрирует применение оператора цикла `For Each` для работы с семейством рабочих листов. Оператор удаляет из рабочей книги рабочий лист **Тест**, если он, конечно, имеется в рабочей книге.

Листинг 4.23. Работа с семейством рабочих листов

```
Sub DemoDeleteSheet()
   Dim ws As Worksheet
   For Each ws In Worksheets
      If ws.Name = "Тест" Then
        ws.Delete
      End If
   Next
End Sub
```

Оператор *While*

Оператор `While` выполняет последовательность инструкций *`statements`*, пока заданное условие *`condition`* возвращает значение `True`.

```
While condition
   [statements]
Wend
```

Оператор While, в отличие от For, повторяется не заданное число раз, а пока выполняется условие. В следующем примере (листинг 4.24) бросается игральная кость до тех пор, пока не выпадет шесть очков. При выпадении шести очков игра заканчивается, и отображается сообщение с указанием, на каком броске она закончилась.

Листинг 4.24. Бросание игральной кости

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

Создание бесконечного цикла оператором *While*

Использование в качестве условия оператора While значения True приводит к созданию бесконечного цикла. Для прерывания его выполнения в теле цикла надо расположить инструкцию, обеспечивающую выход из него при соблюдении некоторого условия.

```
While True
   [statements]
Wend
```

Оператор *Do*

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

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

или

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

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

Примером использования оператора цикла `Do Until` может быть следующий код (листинг 4.25), который обеспечивает повторение цикла до тех пор, пока в поле ввода диалогового окна не будет введен пароль (в данном случае `Winnie`).

Листинг 4.25. Код, ожидающий правильного ввода пароля

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

В качестве примера использования оператора цикла `Do While` приведем следующий код (листинг 4.26), который обеспечивает последовательное отображение имен всех GIF-файлов корневого каталога диска C:. В коде используется функция работы с файлами `Dir`, которая возвращает имя первого подходящего файла. Если подходящего файла нет, то функция `Dir` возвращает пустую строку. При повторном вызове функции `Dir` ее параметр с маской, по которой ищется подходящий файл, опускается.

Листинг 4.26. Последовательное отображение имен файлов с расширением gif

```
Sub DemoShowFiles()
   Dim f As String
   Dim res As String
   f = Dir("C:\*.gif")
   res = f & vbCr
   Do While Len(f)
      f = Dir
      res = res & f & vbCr
```

```
   Loop
   MsgBox res
End Sub
```

Альтернативный выход из цикла

Примером использования альтернативного выхода `Exit Do` из цикла может быть следующий код (листинг 4.27), в котором суммируются вводимые числа. Цикл завершается в случае ввода любого строкового выражения.

Листинг 4.27. Суммирование всех вводимых чисел

```
Sub DemoExitDo()
  Dim s As Double, x As Variant
  s = 0
  Do
      x = InputBox("Введите число")
      If Not IsNumeric(x) Then Exit Do
      s = s + x
  Loop
  MsgBox s
End Sub
```

Создание бесконечного цикла оператором *Do*

Оператор `Do` без условий создает бесконечный цикл. Такие циклы применяются при программировании различных фоновых процессов типа печати документа.

```
Do
Loop
```

Оператор безусловного перехода *GoTo*

Оператор безусловного перехода задает переход на указанную строку внутри процедуры. Обязательный параметр *line* может быть любой меткой строки или номером строки.

```
GoTo line
```

Для использования оператора безусловного перехода надо какой-то строке присвоить метку. Метка должна начинаться с буквы и заканчиваться двоеточием. В качестве примера использования оператора безусловного перехода рассмотрим игру, в которой игроку даны десять попыток броска игральной кости. В случае, если при какой-то из этих попыток выпадает шесть очков, игрок выигрывает, и игра заканчивается (листинг 4.28).

Листинг 4.28. Бросание игральной кости

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

Отметим, что в данном примере использование оператора `GoTo` представляется слишком сложным. Применение цикла `Do` может существенно упростить и сократить код.

Процедура

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

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

- `Private` — ключевое слово, указывающее на то, что процедура является *закрытой*, и областью ее видимости является модуль;
- `Public` — ключевое слово, указывающее на то, что процедура является *открытой* и доступна для всех других процедур во всех модулях;
- `Friend` — ключевое слово, используемое только в модуле класса и указывающее на то, что процедура является *дружественной*, и областью ее видимости является проект;
- `Static` — ключевое слово, указывающее на то, что локальные переменные процедуры сохраняются в промежутках времени между вызовами этой процедуры;

- *name* — имя процедуры, удовлетворяющее стандартным правилам именования переменных;
- *arglist* — список параметров, значения которых передаются в процедуру или возвращаются из процедуры при ее вызове. Разделителем в списке параметров является запятая;
- *statements* — любая группа инструкций, выполняемых в процедуре Sub;
- Exit Sub — оператор, приводящий к немедленному выходу из процедуры.

В следующих примерах (листинги 4.29—4.31) имеются две процедуры DemoSub1 и DemoSub2, расположенные в разных модулях, причем процедура DemoSub2 объявлена как закрытая, и поэтому вызов процедуры DemoSub1 приводит к генерации ошибки.

Листинг 4.29. Процедура DemoSub1 не видит процедуру DemoSub2. Один стандартный модуль

```
Sub DemoSub1()
   DemoSub2
End Sub
```

Листинг 4.30. Процедура DemoSub1 не видит процедуру DemoSub2. Другой стандартный модуль

```
Private Sub DemoSub2()
   Beep
   MsgBox "Вызов принят"
End Sub
```

Для того чтобы избежать этой ошибки, процедура DemoSub2 должна быть объявлена либо как открытая, либо без спецификации области ее видимости.

Листинг 4.31. Процедура DemoSub1 теперь будет видеть процедуру DemoSub2. Другой стандартный модуль

```
Public Sub DemoSub2()
   Beep
   MsgBox "Вызов принят"
End Sub
```

Кроме процедуры Sub, в VBA имеется процедура Function или просто функция.

```
[Public | Private | Friend] [Static] Function name [(arglist)] [As type]
   [statements]
```

```
   [name = expression]
   [Exit Function]
   [statements]
   [name = expression]
End Function
```

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

В следующем коде (листинг 4.32) приводится пример функции `F`, которая находит сумму двух значений.

Листинг 4.32. Пример функции

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

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

Список параметров процедуры

Список параметров процедуры *arglist* имеет следующий синтаксис:

```
[Optional] [ByVal | ByRef] [ParamArray] varname[()] [As type]
↳[= defaultvalue]
```

- `Optional` — ключевое слово, указывающее на то, что параметр не является обязательным. При использовании этого элемента все последующие параметры, которые содержит *arglist*, также должны быть необязательными и их следует описать с помощью ключевого слова `Optional`. Все параметры, описанные как `Optional`, должны иметь тип `Variant`. Не допускается использование ключевого слова `Optional` для любого из параметров, если используется ключевое слово `ParamArray`;

- `ByVal` — ключевое слово, указывающее на то, что этот параметр передается по значению;
- `ByRef` — ключевое слово, указывающее на то, что этот параметр передается по ссылке. Описание `ByRef` используется в VBA по умолчанию;
- `ParamArray` — ключевое слово, которое используется только в качестве последнего элемента в списке *`arglist`* для указания, что конечным параметром является описанный как `Optional` массив значений типа `Variant`. Ключевое слово `ParamArray` позволяет задавать произвольное количество параметров. Оно не может быть использовано с ключевыми словами `ByVal`, `ByRef` или `Optional`;
- *`varname`* — имя переменной, удовлетворяющее стандартным правилам именования переменных;
- *`type`* — тип значений параметра, переданного в процедуру. Допустимые значения: `Byte`, `Boolean`, `Integer`, `Long`, `Currency`, `Single`, `Double`, `Date`, `String` (только строки переменной длины), `Object`, `Variant`. Если отсутствует ключевое слово `Optional`, может быть также указан определяемый пользователем тип или объектный тип;
- *`defaultvalue`* — задает значение, которое параметр принимает по умолчанию. Используется только вместе с параметром `Optional`. Если указан тип `Object`, единственным значением по умолчанию может быть значение `Nothing`.

Организация программы на языке VBA

Программа на языке VBA состоит из одного или нескольких модулей. Обычно текст программы в модуле начинается с директив, которые управляют описанием переменных, способом сравнения строк и т. д. Затем идет объявление переменных и констант уровня модуля или проекта, т. е. переменных и констант, которые можно использовать во всех процедурах либо модуля, либо проекта. Далее располагается код процедур `Sub` и `Function`, составляющих саму программу. Приведем один пример организации модуля (листинг 4.33).

Листинг 4.33. Пример организации модуля

```
Option Base 1
Option Explicit

Private Const Pi = 3.14159265358979

Private Out(2) As Double
```

```
Private Function CircleLength(r As Double) As Double
   CircleLength = 2 * Pi * r
End Function

Private Function AreaDisc(r As Double) As Double
   DiscArea = 4 * Pi * r ^ 2
End Function

Private Sub JointResult(r As Double)
   Out(1) = CircleLength(r)
   Out(2) = DiscArea(r)
   MsgBox Out(1) & vbCr & Out(2)
End Sub

Private Sub ShowResults()
   JointResult 1
End Sub
```

Вызов процедуры и передача значений параметров

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

- Первый способ вызова процедуры `Sub`:

  name arglist

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

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

  ```
  Call name (arglist)
  ```

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

Применяя именованные параметры, VBA позволяет вводить фактические параметры в любом порядке и опускать необязательные (`Optional`). При этом после имени параметра ставятся двоеточие и знак равенства, после которого помещается значение параметра (фактический параметр).

Приводимый ниже код показывает основные способы передачи параметров на примере процедуры `ShowResults` из предыдущего раздела (см. листинг 4.33).

```
Private Sub ShowResults()
   JointResult r:=1
   Dim Rs as Double
   Rs = 2
   JointResult Rs
   Call JointResult(3)
End Sub
```

Процедура с необязательными параметрами

Приведем пример функции с необязательными параметрами (листинг 4.34). Функция `UserName` возвращает строку, состоящую из имени и фамилии, указанных в качестве значений ее параметров. Если какой-то из параметров опущен, то она возвращает неполное имя. При работе с необязательными параметрами необходимо использовать функцию `IsMissing`, возвращающую значение `True`, если соответствующий параметр не был передан в процедуру, и `False` — в противном случае.

Листинг 4.34. Процедура `Function` с необязательными параметрами

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

Sub DemoUserName()
    MsgBox UserName(LastName:="Bond", FirstName:="James")
    '  Bond James
    MsgBox UserName("Bond", "James")
    '  Bond James
    MsgBox UserName("Bond")
    '  Bond
    MsgBox UserName(, "James")
    '  James
    MsgBox UserName()
    ' Nothing
End Sub
```

Специфицирование значений по умолчанию необязательным параметрам

Для необязательного параметра можно специфицировать значение по умолчанию. В следующем примере (листинг 4.35) устанавливаются применяемые по умолчанию интенсивности красной, зеленой и синей составляющих цвета ячеек выделенного диапазона.

Листинг 4.35. Специфицирование значения по умолчанию необязательным параметрам

```
Sub Color(Optional Red As Integer = 0, Optional Green As Integer = 255, _
         Optional Blue As Integer = 0)
   Selection.Interior.Color = RGB(Red, Green, Blue)
End Sub

Sub DemoDefaultValues()
   Color Blue:=175
End Sub
```

Использование неопределенного количества параметров

Как правило, количество передаваемых в процедуру параметров совпадает с количеством определенных у этой процедуры параметров. Однако ключевое слово `ParramArray` предоставляет возможность ввода в процедуру произвольного, заранее не указанного числа параметров (например, как это происходит в функции `СУММ` рабочего листа в MS Excel).

В качестве примера приведем процедуру `Function`, которая из строк, используемых в качестве значений параметров, образует одну строку (листинг 4.36).

Листинг 4.36. Пример процедуры с неопределенным числом параметров

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

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

Использование массива в качестве параметра процедуры

В VBA допустимо использование массива в качестве параметра процедуры. В этом случае массив, используемый в качестве параметра, при описании процедуры надо объявить как динамический. В следующем примере (листинг 4.37) процедура `SumM` возвращает сумму элементов массива, который является ее первым параметром.

Листинг 4.37. Использование массива в качестве параметра процедуры

```
Option Base 1

Sub SumM(ByRef A() As Double, ByRef s As Double)
   Dim x As Variant
   s = 0
   For Each x In A
        s = s + x
   Next
End Sub

Sub DemoSumM()
   Dim B(2, 2) As Double
   Dim s As Double
   B(1, 1) = 1: B(1, 2) = 5
   B(2, 1) = 4: B(2, 2) = 5
   SumM B, s
   MsgBox s
' Результат: 15
End Sub
```

Передача параметров по ссылке и значению

При вызове процедуры вы передаете в нее некоторые параметры. Внутри процедуры этим параметрам могут быть присвоены какие-то значения, которые сохранятся в них после выхода из процедуры. Таким образом, по умолчанию при передаче переменных в качестве параметров, в процедуру передаются физические адреса переменных. Поэтому внутри процедуры может быть модифицировано их содержание. Для явного указания передачи параметров в процедуру по ссылке используется ключевое слово `ByRef`. Другим способом передачи параметров в процедуру является передача их по значению. При этом способе передачи параметра в процедуру попадает не сама переменная, а ее значение. Передача параметра по значению задается ключевым словом `ByVal`. В следующем примере (листинг 4.38) показано отличие передачи параметра по ссылке от передачи параметра по значению.

Листинг 4.38. Передача параметров по ссылке и значению

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

Sub Test()
   a = 1: b = 10: c = 100
   DemoByValByRef a, b, c
   MsgBox a
' Отобразится 1
   MsgBox b
' Отобразится 12
   MsgBox c
' Отобразится 102
End Sub
```

Рекурсивные процедуры

В VBA возможно создание рекурсивных процедур, т. е. процедур, вызывающих самих себя. Классическим примером рекурсивной процедуры является процедура, возвращающая очередной член последовательности чисел Фибоначчи (листинг 4.39). Два первых члена ряда Фибоначчи равны 1, а каждый его последующий член представляет собой сумму двух предыдущих.

Таким образом, *n*-е число Фибоначчи *Fi*(*n*) определяется следующим соотношением:

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

где $Fi(1) = Fi(2) = 1$.

Листинг 4.39. Нахождение *n*-го числа Фибоначчи

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

Другим классическим примером использования рекурсивных функций является нахождение наибольшего общего делителя двух целых чисел по алгоритму Евклида. Наибольший общий делитель (НОД) двух целых чисел — это наибольшее целое, на которое делятся оба числа. Например:

- `НОД(10, 14) = 2`
- `НОД(15, 31) = 1`

Алгоритм Евклида состоит из следующих шагов:

1. Если `a` делится на `b`, то `НОД(a, b) = b`.
2. В противном случае `НОД(a, b) = НОД(b, a Mod b)`.

Приводимая ниже рекурсивная функция (листинг 4.40) программирует алгоритм Евклида.

Листинг 4.40. Нахождение наибольшего общего делителя

```
Function НОД(Целое1 As Long, Целое2 As Long) As Long
   If Целое2 Mod Целое1 = 0 Then
      НОД = Целое1
   Else
      НОД = НОД(Целое2, Целое1 Mod Целое2)
   End If
End Function
```

Фракталы

Еще одним интересным примером рекурсивных процедур является построение фракталов. Фракталы состоят из последовательности геометрических объектов, причем первоначальный объект рисуется в большом масштабе.

В последующем к этому объекту добавляются его уменьшенные копии. При этом копии могут иметь как ту же самую ориентацию, что и первоначальный объект, так и измененную ориентацию. Конечно, в VBA, в отличие от Visual Basic, отсутствуют средства графических построений непосредственно в форме. Зато, благодаря семейству объектов `Shapes`, они в избытке имеются для создания графики на рабочем листе. В приводимом ниже примере (листинг 4.41) мы просто покроем рабочий лист причудливым геометрическим узором при помощи фракталов. Данный фрактал имеет простую структуру. Первоначально рисуется большой квадрат, а потом к его углам пристраиваются его уменьшенные копии и так до тех пор, пока пристраиваемые квадраты не станут достаточно маленькими (рис. 4.7). Для того чтобы увидеть картинку на рабочем листе, уменьшите масштаб при помощи команды **Вид | Масштаб** до 10%.

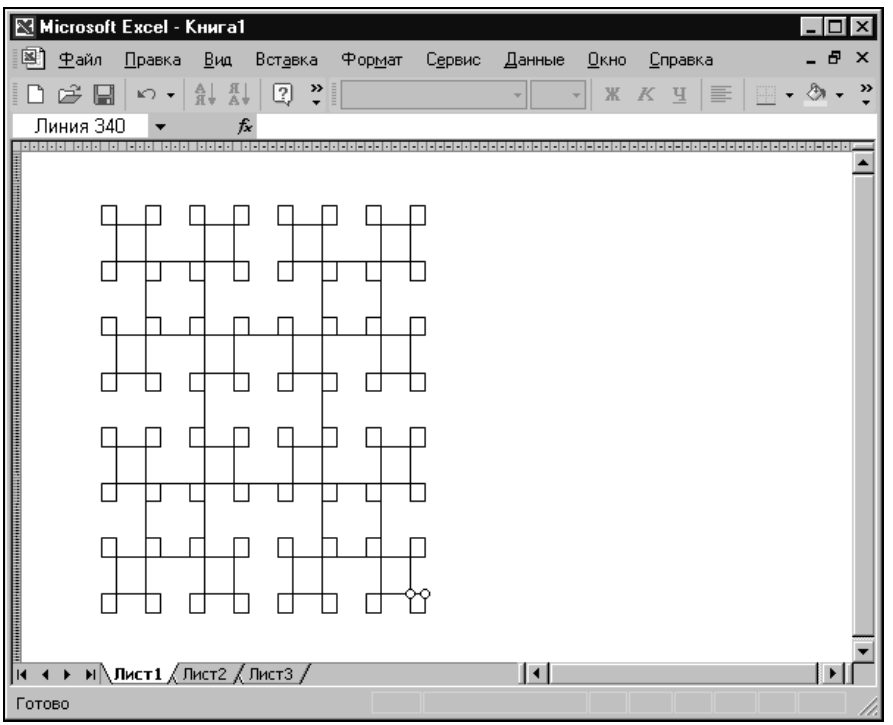


Рис. 4.7. Пример построения фракталов на рабочем листе

Листинг 4.41. Построение фракталов на основе прямоугольников

```
Sub Square(x As Long, y As Long, Size As Long)
   If Size < 80 Then Exit Sub
```

```
   ActiveSheet.Shapes.AddLine(x, y, x, y + Size).Select
   ActiveSheet.Shapes.AddLine(x, y + Size, _
                              x + Size, y + Size).Select
   ActiveSheet.Shapes.AddLine(x + Size, y + Size, _
                              x + Size, y).Select
   ActiveSheet.Shapes.AddLine(x + Size, y, x, y).Select

   Square x - Size / 2, y - Size / 2, Size / 2
   Square x + Size, y - Size / 2, Size / 2
   Square x - Size / 2, y + Size, Size / 2
   Square x + Size, y + Size, Size / 2
End Sub

Sub DemoSquare()
   Square 1000, 800, 640
End Sub
```

Приведем еще один пример построения фракталов. В нем фракталы строятся на основе правильных многоугольников, в частности, на рис. 4.8 показан результат действия программы при построении фракталов на основе пятиугольных звезд (листинг 4.42). В этой программе имеется одна особенность, на которую стоит обратить внимание. А именно на применение пользовательского типа для работы с точками плоскости. Пользовательский тип в данном случае обеспечивает дополнительную прозрачность кода.

Листинг 4.42. Построение фракталов на основе пятиконечных звезд

```
Option Explicit

Private Pi As Double

Private Type Point
    x As Long
    y As Long
End Type

Sub ShapeBuilder(x As Long, y As Long, Size As Long, n As Long)
   Dim points(1 To 10) As Point
   Dim i As Integer
   If Size < 10 Then Exit Sub
   For i = 0 To n - 1
      With points(i + 1)
        .x = x + Size * Cos(2 * Pi * i / n - Pi / 2)
        .y = y + Size * Sin(2 * Pi * i / n - Pi / 2)
      End With
   Next
```

```
   For i = 1 To n
      With points((i + 1) Mod n + 1)
         ActiveSheet.Shapes.AddLine(.x, .y, _
                                    points(i).x, points(i).y).Select
       End With
   Next

   For i = 1 To n
      With points(i)
         ShapeBuilder .x, .y, Size / 2.5, n
      End With
   Next
End Sub

Private Sub DemoShapeBuilder()
    Pi = Application.WorksheetFunction.Pi()
    ShapeBuilder 500, 500, 300, 5
End Sub
```

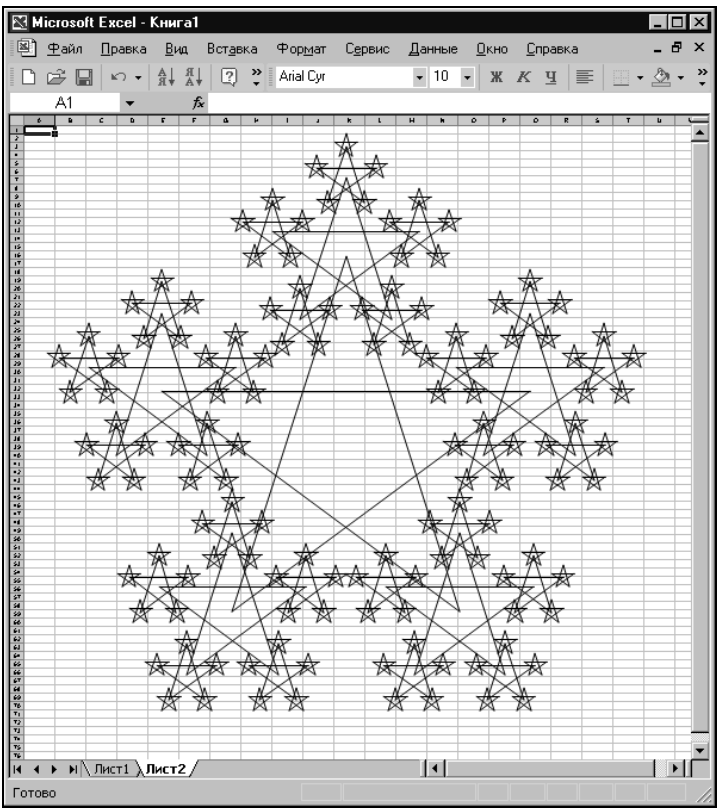


Рис. 4.8. Пример построения фракталов на основе звезд

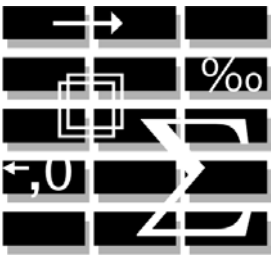
Используя рекурсию, будьте осторожны

Несмотря на элегантность рекурсивных процедур, применять их надо с осторожностью, т. к. неаккуратное использование может привести к проблемам с памятью — многократный вызов такой процедуры быстро исчерпывает стековую память. Кроме того, программы с рекурсией выполняются существенно дольше, чем программы, использующие циклы, при решении тех же задач. В этом вы сами можете легко убедиться, сравнив время, за которое находится, например, тридцатое число Фибоначчи с использованием рекурсивной функции из листинга 4.39 и при помощи следующей программы (листинг 4.43).

Листинг 4.43. Нахождение числа Фибоначчи без рекурсии

```
Sub Fibonacci()
   Dim n As Long, i As Long
   Dim f1 As Long, f2 As Long
   n = 30
   f1 = 1 : f2 = 2
   For i = 3 To n
      f1 = f1 + f2
      f1 = f2 – f1
   Next
   MsgBox f2
End Sub
```

ГЛАВА 5



Классы, объекты и семейства

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

Классы и экземпляры

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

- *Полем* называется переменная, содержащая некоторое значение. Таким образом, поле предоставляет информацию об объекте. Например, если имеется объект `Car` (автомобиль), то его полем может быть поле `Cylinders` (цилиндры), хранящее информацию о числе цилиндров в двигателе автомобиля.
- *Метод* — это код, программирующий некоторые действия, которые должен совершить объект. Например, объект `Car` может быть перекрашен методом `Repaint`.
- *Свойство* играет ту же роль, что и поле, а именно предоставляет информацию об объекте, но при его создании используются специальные про-

цедуры `Property`, которые предоставляют больше возможностей как по установке значения, так и его возвращения. Свойство позволяет отделить данные от объекта, сделав их более устойчивыми.

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

Объявление класса

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

Опишем процесс создания класса в виде следующей последовательности шагов:

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

В качестве примера создадим класс `Point`, моделирующий точку плоскости (листинг 5.1). В этом классе имеются только два поля `x` и `y`, которые задают координаты точки. Модификатор доступа `Public` устанавливает, что поля доступны для всех внешних кодов и могут быть опрошены и изменены любым из них.

Листинг 5.1. Модуль класса `Point`

```
Public x As Integer ' Поле x
Public y As Integer ' Поле y
```

Создание экземпляра класса

Для того чтобы использовать созданный класс, нужно иметь возможность получения *экземпляра* этого класса. Экземпляр — это объект, имеющий тип данного класса. Для любого созданного класса можно получить экземпляры так же, как для любого другого типа данных, но при этом при объявлении объекта надо указать ключевое слово `New`. После создания экземпляра класса у него можно считывать или устанавливать значения полей, свойств, применять к нему методы. Например, в следующем коде (листинг 5.2) создается экземпляр класса `Point`, у которого устанавливаются значения полей `x` и `y`. Кроме того, некоторая переменная может быть сначала объявлена как объектная (в данном случае типа `Point`), а затем при помощи оператора присваивания `Set` ей уже может быть назначено значение.

Листинг 5.2. Создание экземпляра класса. Стандартный модуль

```
Sub TestPoint()
   Dim p1 As New Point
   p1.x = 2
   p1.y = 3
   Dim p2 As Point
   Set p2 = p1
   Debug.Print p1.x & vbTab & p1.y
   Debug.Print p2.x & vbTab & p2.y
End Sub
```

Инициализация значений полей

Начальные значения полей экземпляра класса можно устанавливать в зависимости от бизнес-логики проекта. Для этого достаточно в процедуре обработки события `Initialize` модуля класса полям присвоить требуемые значения. Например, следующая модификация класса `Point` (листинг 5.3) обеспечит то, что все создаваемые экземпляры этого класса являются точками (1, 1), а не (0, 0), как это было раньше.

Листинг 5.3, *а*. Модуль класса `Point`

```
Public x As Integer
Public y As Integer

Private Sub Class_Initialize()
   x = 1
   y = 1
End Sub
```

Листинг 5.3, *б*. Создание экземпляра класса. Стандартный модуль

```
Sub TestInitPoint()
   Dim p As New Point
   Debug.Print p.x & vbTab & p.y
End Sub
```

Ключевое слово *Me*

Ключевое слово `Me` возвращает экземпляр данного класса, через который можно обращаться к полям. Поэтому код инициализации значений полей класса `Point` можно оформить следующим равносильным способом (листинг 5.4).

Листинг 5.4. Класс `Point` с ключевым словом `Me`

```
Public x As Integer
Public y As Integer

Private Sub Class_Initialize()
   Me.x = 1
   Me.y = 1
End Sub
```

Ключевое слово *Nothing* и удаление объекта из памяти

Для удаления ссылки из объектной переменной используется ключевое слово `Nothing`. Например:

```
Dim p As New Point
p.x = 1
Set p = Nothing
```

Если на используемый объект не ссылаются другие переменные, то Windows удаляет его из памяти. Обратите внимание на то, что значение `Nothing` должно присваиваться посредством оператора `Set`, как при любом присваивании объектов.

Методы

Методы класса обычно содержат код, который анализирует состояние объекта и изменяет его. Например, классы часто обладают определенными функциями, которые не сводятся к простому чтению или установке значе-

ний некоторых параметров, а требуют проведения определенных вычислений. В этом случае на помощь и приходят методы. По своей сути методы представляют собой процедуры. В том случае, если они возвращают или устанавливают значения, их удобнее программировать в виде функций.

Вызов метода представляет собой операцию, выполняемую с объектом посредством ссылки на него, затем указания точки, после которой идет имя метода, за которым в скобках перечисляется список фактических параметров.

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

В следующем коде (листинг 5.5) расширяются функциональные возможности класса `Point` путем добавления в него трех методов. Метод `Move` перемещает точку в направлении, заданном другой точкой. Метод `Length` возвращает длину вектора, т. е. расстояние от начала координат до точки. Метод `ToString` аккумулирует информацию об экземпляре класса в виде строки. Первый из этих методов реализован в виде процедуры, а второй и третий — в виде функций.

Листинг 5.5, *а*. Методы. Модуль класса `Point`

```
Public x As Integer
Public y As Integer

Public Sub Move(ByVal pt As Point)
   x = x + pt.x
   y = y + pt.y
End Sub

Public Function Length() As Double
   Length = Sqr(x ^ 2 + y ^ 2)
End Function

Public Function ToString() As String
   ToString = "(" & x & "," & y & ")"
End Function
```

Листинг 5.5, *б*. Методы. Стандартный модуль

```
Sub TestPointWithMethods()
   Dim p1 As New Point
   p1.x = 1
   p1.y = 1
   Debug.Print p1.ToString
   Debug.Print p1.Length
```

```
    Dim p2 As New Point
    p2.x = 2
    p2.y = 2
    Debug.Print p2.ToString
    p1.Move p2
    Debug.Print p1.ToString
    Debug.Print p1.Length
End Sub
```

Свойства как средство ограничения доступа к полям класса

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

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

В приводимом ниже коде (листинг 5.6) создается класс `Employee`, инкапсулирующий информацию о сотруднике фирмы. В этом классе имеются три закрытых поля: `fname`, `lname` и `pos`, которые описывают имя, фамилию и должность сотрудника. Для установки и считывания значений этих полей в классе определены три свойства: `FirstName`, `LastName` и `Position`. Также в классе описан метод `ToDebug`, который выводит комбинированную информацию об экземпляре класса в окно **Immediate**.

Листинг 5.6, *а*. Свойства. Модуль класса `Employee`

```
Private fname As String
Private lname As String
Private pos As String
```

```
Property Get FirstName() As String
   FirstName = fname
End Property

Property Let FirstName(ByRef newfname As String)
   fname = newfname
End Property

Property Get LastName() As String
   LastName = lname
End Property

Property Let LastName(ByRef newlname As String)
   lname = newlname
End Property

Property Get Position() As String
   Position = pos
End Property

Property Let Position(ByRef newpos As String)
   pos = newpos
End Property

Public Sub ToDebug()
   Debug.Print "First Name: " & fname & vbCr & _
               "Last Name: " & lname & vbCr & _
               "Position: " & pos
End Sub
```

Листинг 5.6, *б*. Свойства. Стандартный модуль

```
Sub DemoEmployee()
   Dim emp1 As New Employee
   emp1.FirstName = "James"
   emp1.LastName = "Bond"
   emp1.Position = "Secret agent 007"
   emp1.ToDebug
   Dim emp2 As New Employee
   emp2.FirstName = "Winnie"
   emp2.LastName = "Pooh"
   emp2.Position = "Bear"
   emp2.ToDebug
End Sub
```

Свойства "только для чтения" и "только для записи"

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

В следующем примере (листинг 5.7) создается класс `Point`. Свойства `GetX` и `GetY` являются свойствами "только для чтения" и позволяют считывать значения полей. Свойства же `SetX` и `SetY` являются свойствами "только для записи" и предназначены для задания значений полям. Кроме того, в классе объявлен метод `ToString` для вывода общей информации об экземпляре класса в виде строки.

Листинг 5.7, *а*. Свойства "только для чтения" и "только для записи". Модуль класса `Point`

```
Private x As Integer
Private y As Integer

Public Property Get GetX() As String
   GetX = x
End Property

Public Property Let SetX(ByRef valX As String)
   x = valX
End Property

Public Property Get GetY() As String
   GetY = y
End Property

Public Property Let SetY(ByRef valY As String)
   y = valY
End Property

Public Function ToString() As String
   ToString = "(" & x & "," & y & ")"
End Function
```

Листинг 5.7, *б*. Свойства "только для чтения" и "только для записи". Стандартный модуль

```
Sub DemoOnlyProperties()
   Dim p As New Point
   p.SetX = 1
   p.SetY = 4
   Debug.Print p.ToString
   Debug.Print p.GetX & vbTab & p.GetY
End Sub
```

События

До сих пор объекты, которые мы создавали, были глухими и немыми, т. е. они как не получали сообщений от других объектов, так и не передавали их, что может навести на мысль, что в VBA программы представляют собой последовательность выполняемых друг за другом процедур. Это совсем не так. Зачастую момент выполнения процедуры в программе обусловлен внешним фактором — *событием* (event). Например, нажатием кнопки, выбором переключателя, сворачиванием формы и т. д. Таким образом, событие играет в приложении роль сигнала, информирующего о том, что в системе произошло что-то важное, требующее дополнительного внимания.

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

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

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

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

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

При объявлении переменной, которой будет присвоено значение экземпляра того класса, который может генерировать событие, надо дополнительно

добавить ключевое слово `WithEvents`. Например, в приводимом ниже коде событие будет генерироваться у переменной `sc` типа `SumListener`.

```
Private WithEvents sc As SumListener
```

Для того чтобы отреагировать на событие, надо его ассоциировать со специальной процедурой, которая называется *обработчиком события* (event handlers). Такое ассоциирование производится с использованием шаблона процедуры обработки события.

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

В приводимом ниже примере это будет следующая процедура.

```
Sub sc_SumDone(ByVal s As Double)
   Debug.Print CStr("Sum done " & s)
End Sub
```

Рассмотрим простой пример (листинг 5.8). В классе `SumListener` имеются два открытых поля `a` и `b` и закрытое — `s`. Метод `Sum` суммирует значения из полей `a` и `b` и помещает результат в поле `s`. Метод `GetRes` возвращает значение найденной суммы. При выполнении метода `Sum` по завершении вычисления суммы генерируется событие `SumDone(ByVal s As Double)`, параметр которого возвращает значение суммы. Таким образом, в процедуре `DemoEvent` результат в окно **Immediate** сначала выводится за счет обработки события `SumDone`, которое генерируется при использовании метода `Sum`, а уж затем результат повторно явным образом выводится методом `GetRes`.

Листинг 5.8, *а*. Класс с событием. Модуль класса `SumListener`

```
Public a As Double
Public b As Double
Private s As Double

Public Event SumDone(ByVal s As Double)

Public Sub Sum()
   s = a + b
   RaiseEvent SumDone(s)
End Sub

Public Function GetRes() As Double
   GetRes = s
End Function
```

Листинг 5.8, *б*. Класс с событием. Модуль листа или `ЭтаКнига`

```
Private WithEvents sc As SumListener

Sub DemoEvent()
   Set sc = New SumListener
   sc.a = 1
   sc.b = 3
   sc.Sum
   Debug.Print sc.GetRes()
End Sub

Sub sc_SumDone(ByVal s As Double)
   Debug.Print CStr("Sum done " & s)
End Sub
```

Таймер как пример класса, генерирующего события

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

Для испытания этого класса создайте форму, в которой расположите два поля ввода и две кнопки (**Start** и **Stop**), а в модуле формы наберите приводимый ниже код (листинг 5.9). Нажатие кнопки **Start** обеспечит включение таймера на 5 с. В первое поле ввода будет выводиться число прошедших с момента включения 1/100 долей секунды. Во второе поле ввода — сообщение о том, что либо таймер еще работает, либо прекратил работу, если такое событие случилось. Нажатие кнопки **Stop** вызовет остановку работы таймера.

Листинг 5.9, *а*. Таймер. Модуль класса `TimerState`

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

Public Sub TimerTask()
    Dim startTime As Double
```

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

Листинг 5.9, *б*. Таймер. Модуль формы

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

Private Sub UserForm_Initialize()
   Me.Caption = "Timer"
   CommandButton1.Caption = "Start"
   CommandButton2.Caption = "Stop"
   Set ts = New TimerState
End Sub

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

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

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

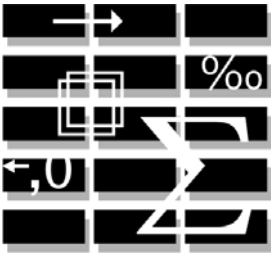
```
Private Sub ts_Collapse()
   TextBox1.Text = ""
   TextBox2.Text = "End"
   i = 0
   DoEvents
End Sub
```

Объект *Collection*

Объект `Collection` позволяет динамически группировать семейство объектов, идентифицированных по индексу. Объект `Collection` имеет лишь одно свойство `Count`, возвращающее число элементов набора, и три метода, приведенные в табл. 5.1.

Таблица 5.1. *Методы объекта* `Collection`

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



ГЛАВА 6

Элементы управления на рабочем листе

Элементы управления, размещенные на рабочем листе, создают индивидуальный интерфейс проекта, внедренный непосредственно в рабочий лист, т. е. интерфейс, который максимально приближен к пользователю. Они позволяют автоматизировать и алгоритмизировать работу пользователя, тем самым упрощая ее и повышая эффективность работы пользователя. Элементы управления обеспечивают создание защиты "от дурака", т. е. с их помощью можно достичь того, чтобы пользователь мог действовать только в рамках бизнес-логики проекта, не нарушая его целостности.

Панель инструментов *Элементы управления*

На рабочем листе при помощи панели инструментов **Элементы управления**, которая отображается на экране выбором команды **Вид | Панели инструментов | Элементы управления** (рис. 6.1) можно размещать различные элементы управления. В табл. 6.1 приведен список основных элементов управления и соответствующих кнопок панели инструментов.

Примечание

Сразу же стоить отметить, что в MS Excel существуют две похожие панели инструментов **Элементы управления** и **Формы**. Панель инструментов **Формы** унаследована от версии MS Excel 5.0 и в основном используется для совместимости приложений, созданных в старых версиях.

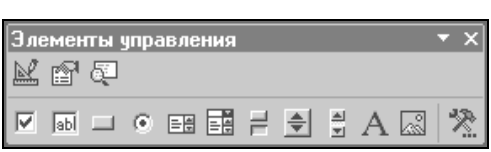


Рис. 6.1. Панель инструментов для рабочего листа

Таблица 6.1. *Элементы управления из панели инструментов*

Элемент управления	Имя	Префикс	Кнопка, его создающая
Поле	TextBox	txt	
Надпись	Label	lbl	
Кнопка	CommandButton	cmd	
Список	ListBox	lst	
Поле со списком	ComboBox	cbo	
Полоса прокрутки	ScrollBar	scr	
Счетчик	SpinButton	spn	
Переключатель	OptionButton	opt	
Флажок	CheckBox	chk	
Выключатель	ToggleButton	tgl	
Рисунок	Image	img	

Размещение элемента управления на рабочем листе и написание кода

Элементы управления на рабочем листе можно размещать тогда, когда MS Excel находится в режиме конструктора элементов управления. Для того чтобы перевести MS Excel в режим конструктора, надо нажать кнопку **Режим конструктора**  панели инструментов **Элементы управления**.

После перевода MS Excel в режим конструктора можно начать непосредственное создание элементов управления на рабочем листе.

Нажмите соответствующую кнопку панели инструментов **Элементы управления** и с помощью мыши перетащите рамку элемента управления в искомое местоположение.

Используя маркеры перемещения и изменения размера, элемент управления можно перемещать, изменять его размеры. Кроме того, элементы управления можно копировать в буфер обмена, вставлять их из буфера обмена и удалять с рабочего листа.

Совет

Как и любой графический объект, элемент управления можно рисовать при нажатой клавише <Shift>, что обеспечит ему пропорциональное изменение размеров, либо при нажатой клавише <Alt> — для привязки элемента управления к сетке рабочего листа.

Элементы управления являются объектами, т. е. они, как и любые другие объекты, обладают свойствами, методами и событиями. Значения свойств элементов управления устанавливаются как в коде, так и на этапе их конструирования. Для того чтобы установить значения свойств на этапе конструирования, надо выбрать элемент управления и нажать кнопку **Свойства**  панели инструментов **Элементы управления**. На экране отобразится окно **Properties** (рис. 6.2). В левой половине этого окна перечисляются свойства элемента, а в правой — имеются либо поля ввода, либо раскрывающиеся списки для установки значений этих свойств.

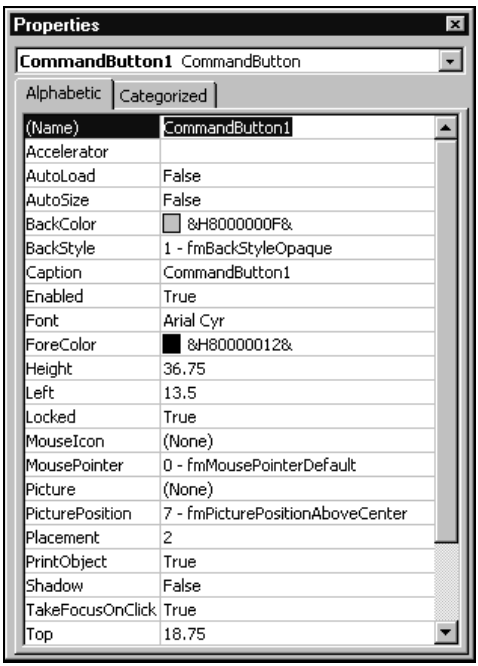


Рис. 6.2. Окно **Properties**

Код процедуры, обрабатывающий события, связанные с элементом управления, набирается в модуле рабочего листа, на котором расположен элемент управления. Для перехода в этот модуль выберите элемент управления и нажмите кнопку **Исходный текст**  панели инструментов **Элементы управления**.

По завершении конструирования элемента управления не забудьте выйти из режима конструирования, повторно нажав кнопку **Режим конструктора** .

Ваш первый проект с элементом управления

Построим ваш первый проект с элементом управления. Следуя традициям, в этом проекте на рабочем листе расположим кнопку, нажатие на которую приведет к отображению на экране диалогового окна с приветствием "Hello, World!".

1. Нажмите кнопку **Режим конструктора**  панели инструментов **Элементы управления** для того, чтобы перевести MS Excel в режим конструктора элементов управления.
2. Нажмите кнопку  панели инструментов **Элементы управления**. Нарисуйте на рабочем листе элемент управления **Кнопка**, точно так же, как и при помощи любого графического редактора, в том месте рабочего листа, где вы задумали расположить кнопку.

Примечание

На поверхности первого созданного элемента управления **Кнопка** автоматически будет отображена надпись **CommandButton1**. Если вы создадите второй элемент управления **Кнопка**, то на его поверхности отобразится надпись **CommandButton2** и т. д. Текст, отображаемый на поверхности элемента управления, устанавливается значением свойства `Caption`. Кроме того, VBA устанавливает значение свойства `Name` объекта (т. е. его имя), используемое по умолчанию. Для первой созданной кнопки значение свойства `Name` устанавливается VBA равным `CommandButton1`, для второй — `CommandButton2` и т. д.

3. Выберите созданную кнопку. Нажмите кнопку **Свойства**  панели инструментов **Элементы управления**. На экране отобразится окно **Properties**. Установите в этом окне значение свойства `Name` равным `cmdHello` вместо безликого `CommandButton1`. Установите значения свойства `Caption` равным `Hello`.

Совет

Установка осмысленного значения свойства `Name` элемента управления упрощает чтение кода, т. к. помогает по имени элемента управления распознать функцию, выполняемую им в проекте. Префикс же в имени (в нашем случае `cmd`) используется также для удобства. Он идентифицирует тип элемента управления (см. табл. 9.4).

4. Выберите созданную кнопку. Нажмите кнопку **Исходный текст**  панели инструментов **Элементы управления**. В результате откроется редактор

Visual Basic, причем в редакторе кода автоматически будет создана первая и последняя инструкции обработки события `Click` кнопки, генерируемого при ее нажатии (листинг 6.1, *а*).

Листинг 6.1, *а*. "Hello, World!"

```
Private Sub cmdHello_Click()
End Sub
```

5. В процедуру обработки события `Click` кнопки добавьте инструкцию, которая отобразит на экране диалоговое окно с приветствием (листинг 6.1, *б*).

Листинг 6.1, *б*. "Hello, World!"

```
Private Sub cmdHello_Click()
   Msgbox "Hello, World!", vbExclamation
End Sub
```

6. Нажмите кнопку **Режим конструктора** панели инструментов **Элементы управления** для того, чтобы выйти из режима конструирования.

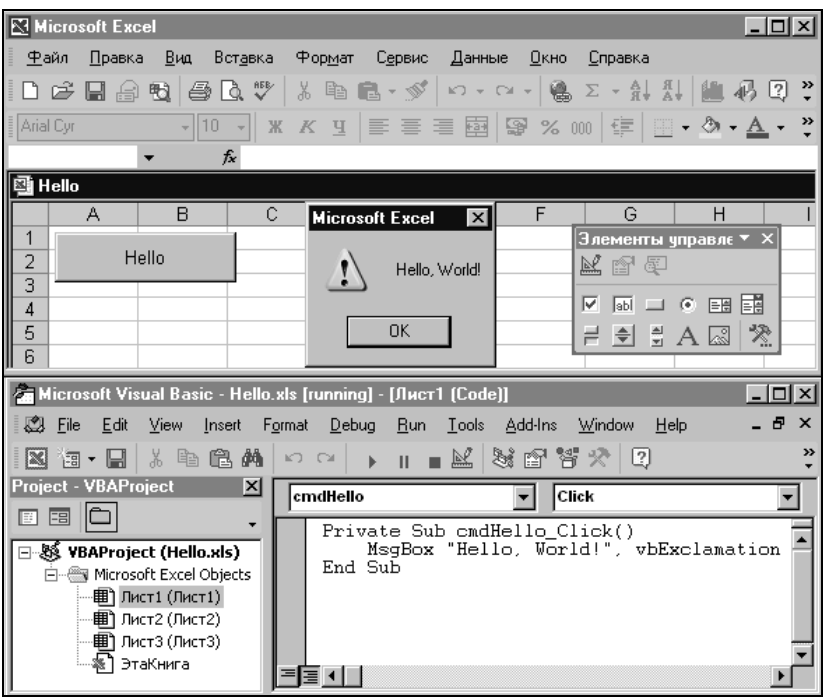


Рис. 6.3. Проект "Hello, World!". Рабочий лист и редактор Visual Basic

Проект готов. Теперь можно и протестировать созданную кнопку **Hello**. Нажмите ее, и если на экране отобразится диалоговое окно с приветствием, то вы сделали все правильно (рис. 6.3).

Общие свойства элементов управления

Элементы управления обладают большой коллекцией свойств, позволяющих устанавливать различные параметры объекта от его местоположения и размеров до отображаемого в нем текста и рисунка. В табл. 6.2 перечислены общие свойства элементов управления.

Таблица 6.2. *Общие свойства элементов управления*

Свойство	Описание
`AutoSize`	Специфицирует, надо ли автоматически изменять размеры элемента управления с тем, чтобы в нем умещалась вся выводимая информация
`BackColor`	Задает цвет фона
`BackStyle`	Устанавливает прозрачность фона. Допустимыми значениями являются следующие константы: `fmBackStyleTransparent` и `fmBackStyleOpaque`
`BottomRightCell`, `TopLeftCell`	Возвращают ссылки на ячейки, расположенные под левым верхним и правым нижним углами элемента управления
`Caption`	Задает строку текста, отображаемую на элементе управления
`ControlTipText`	Возвращает текст всплывающей подсказки
`Enabled`	Устанавливает, достижим ли для пользователя элемент управления
`Font`	Возвращает объект `Font` для задания параметров шрифта
`ForeColor`	Задает цвет шрифта
`Height`, `Width`	Устанавливают высоту и ширину элемента управления
`Left`, `Top`	Координаты левого верхнего угла элемента управления
`MouseIcon`	Назначает пользовательский указатель мыши
`MousePointer`	Специфицирует тип указателя мыши
`Name`	Назначает имя объекта
`OldHeight`, `OldWidth`	Возвращают высоту и ширину элемента управления **Кнопка** до изменения его размеров
`OldLeft`, `OldTop`	Возвращают координаты левого верхнего угла кнопки до ее перемещения

Таблица 6.2 *(окончание)*

Свойство	Описание
`Parent`	Специфицирует ссылку на объект-контейнер для данного элемента управления
`Picture`	Задает ссылку на файл с растровым изображением, используемым в качестве фона элемента управления
`PicturePosition`	Устанавливает местоположение рисунка по отношению к надписи. Допустимыми значениями являются следующие константы: `fmPicturePositionLeftTop`, `fmPicturePositionLeftCenter`, `fmPicturePositionLeftBottom`, `fmPicturePositionRightTop`, `fmPicturePositionRightCenter`, `fmPicturePositionRightBottom`, `fmPicturePositionAboveLeft`, `fmPicturePositionAboveCenter`, `fmPicturePositionAboveRight`, `fmPicturePositionBelowLeft`, `fmPicturePositionBelowCenter`, `fmPicturePositionBelowRight`, `fmPicturePositionCenter`
`PrintObject`	Определяет, надо ли выводить элемент управления при печати
`Tag`	Задает параметр, используемый для идентификации конкретного элемента управления
`TakeFocusOnClick`	Устанавливает, получает ли элемент управления фокус после щелчка на нем
`Visible`	Задает видимость элемента управления
`WordWrap`	Устанавливает, надо ли переносить слова, если они не помещаются в строке, отображаемой в элементе управления

Общие методы элементов управления

Элементы управления обладают несколькими общими методами, позволяющими перемещать, располагать их и управлять фокусом. В табл. 6.3 перечислены эти методы.

Таблица 6.3. *Общие методы элементов управления*

Методы	Описание
`Move`	Перемещает элемент управления
`SetFocus`	Устанавливает фокус на элементе управления
`BringToFront`, `SendToBack`	Располагают элемент управления на переднем или заднем плане

Таблица 6.3 *(окончание)*

Методы	Описание
`ZOrder`	Располагает элемент управления на переднем или заднем плане по отношению к другим элементам управления в зависимости от значения параметра, допустимыми значениями которого являются следующие константы: `fmTop` и `fmBottom`

Общие события элементов управления

Элементы управления имеют большую коллекцию процедур, способных перехватывать и обрабатывать различные события, происходящие в системе, и действия, произведенные пользователем, от щелчка мышью до обнаружения ошибки. В табл. 6.4 перечислены общие для элементов управления события.

Таблица 6.4. *Общие события элементов управления*

Событие	Описание
`BeforeDragOver`	Происходит при буксировке данных
`BeforeDropOrPaste`	Происходит перед вставкой данных, производимой буксировкой
`Click`	Происходит, когда пользователь щелкает на элементе управления
`DblClick`	Происходит, когда пользователь дважды щелкает мышью на элементе управления
`Enter`, `Exit`	Происходят, когда элемент управления получает или теряет фокус
`Error`	Происходит, когда элемент управления обнаружил ошибку, но не может передать сообщение
`KeyDown`, `KeyUp`	Происходят, когда пользователь нажимает и отпускает любую клавишу на клавиатуре, а элемент управления имеет фокус
`KeyPress`	Происходит, когда пользователь нажимает любую клавишу на клавиатуре, кроме функциональных клавиш, клавиш управлением курсором и клавиш управления, а элемент управления имеет фокус
`MouseDown`, `MouseUp`	Происходят, когда пользователь нажимает и отпускает любую кнопку мыши
`MouseMove`	Происходит, когда пользователь передвигает указатель мыши над элементом управления

Кнопка (*CommandButton*)

Элемент управления **Кнопка** (CommandButton) в основном используется для инициирования выполнения некоторых действий, вызываемых нажатием кнопки, например запуск программы или остановка ее выполнения, печать результатов и т. д. Таким образом, основным событием, связанным с кнопкой, является событие `Click`. Основным свойством кнопки является свойство `Caption`, возвращающее и устанавливающее текст, отображаемый на поверхности кнопки.

Кнопочное меню

Рассмотрим бизнес-ситуацию создания кнопочного меню. В книге имеются три рабочих листа. На первом из них расположены две кнопки с именами двух других листов. Нажатие кнопки приводит к активизации одноименного листа, причем при нажатии второй кнопки производится не только активизация, но и прокрутка листа с тем, чтобы указанная ячейка отображалась в левом верхнем углу окна листа.

Итак, создайте рабочую книгу с листами **Лист1**, **Лист2** и **Лист3**. На рабочем листе **Лист1** расположите две кнопки (рис. 6.4). При помощи окна **Properties** установите им значения свойств, как показано в табл. 6.5.

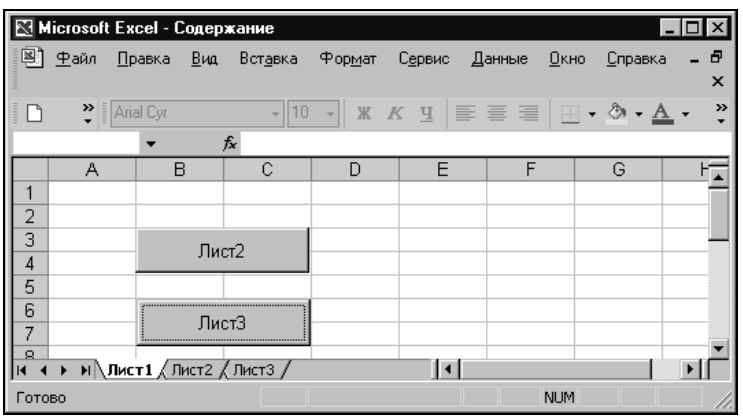


Рис. 6.4. Кнопочное меню

Таблица 6.5. *Значения свойств, установленные в окне* ***Properties***

Объект	Свойство	Значение
Кнопка	`Name`	`cmdSheet2`
	`Caption`	`Лист2`

Таблица 6.5 (окончание)

Объект	Свойство	Значение
Кнопка	Name	cmdSheet3
	Caption	Лист3

В модуле рабочего листа **Лист1** наберите следующий код (листинг 6.2). Проект готов. Активизация листа производится, конечно, методом Activate объекта Worksheet. Прокрутка же листа с тем, чтобы указанная ячейка (в данном случае **T30**) отображалась в левом верхнем углу окна листа, осуществляется свойствами ScrollColumn и ScrollRow объекта Window.

Листинг 6.2. Автоматическое содержание книги. Модуль рабочего листа Лист1

```
Private Sub cmdSheet2_Click()
   Sheets("Лист2").Activate
End Sub

Private Sub cmdSheet3_Click()
   Sheets("Лист3").Activate
   ActiveWindow.ScrollColumn = 20
   ActiveWindow.ScrollRow = 30
End Sub
```

Примечание

Для прокрутки экрана с тем, чтобы отобразился искомый диапазон, можно также применить метод Show объекта Range. Например, Cells(30, 20).Show.

Навигация по книге при помощи гиперссылок

Конечно, навигацию по книге можно производить при помощи кнопочного меню, но можно и классическим способом, без написания кода, средствами гиперссылок. Итак, в рабочей книге с тремя рабочими листами **Лист1**, **Лист2** и **Лист3**:

1. Выберите лист **Лист1**.
2. В ячейку **B1** введите Лист2.
3. Выберите команду **Вставка | Гиперссылка**. На экране отобразится окно **Добавление гиперссылки**. Переключатели **файлом**, **веб-страницей**, **местом в документе**, **новым документом**, **электронной почтой** группы **Связать с** указывают на документ, в который будет дана ссылка в гиперссылке.

4. В данном случае выберите переключатель **местом в документе**. Поле **Текст** задает текст всплывающей подсказки. В поле **Введите адрес ячейки** дается ссылка на ячейку в документе, в которую происходит переход. В списке **Или выберите место в документе** приводится список листов и именованных объектов книги.
5. В данном случае выберите **Лист2**. Нажмите кнопку **OK**.
6. В ячейку **B2** введите `Лист3`. Выберите команду **Вставка | Гиперссылка**. На экране отобразится окно **Добавление гиперссылки** (рис. 6.5).
7. Выберите переключатель **местом в документе** группы **Связать с**. В поле **Введите адрес ячейки** введите `T30`. В списке **Или выберите место в документе** выберите **Лист3**. Нажмите кнопку **OK**.

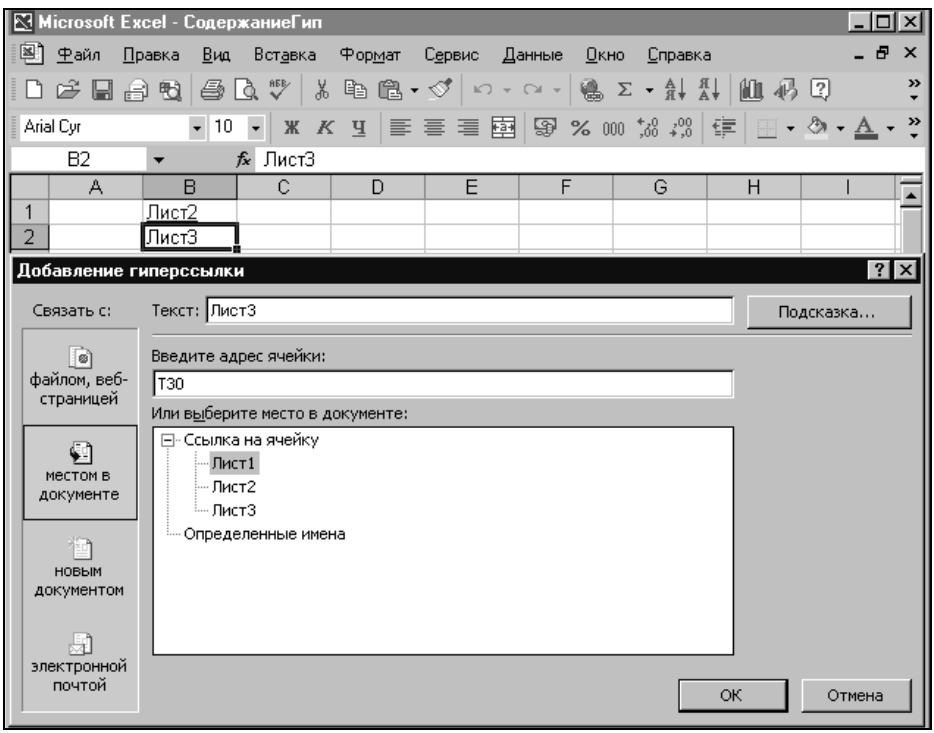


Рис. 6.5. Окно **Добавление гиперссылки**

Кнопочный сценарий

Кнопки позволяют производить не только навигацию, но и вводить данные в ячейки, тем самым создавая кнопочные сценарии, например, если необходимо отобразить итоговую сумму, скажем, расходов или прибыли при

различных сочетаниях ее компонентов. Абстрагируясь, рассмотрим задачу нахождения итоговой суммы $a + b + c$ трех переменных при двух возможных вариантах их значений, приведенных в табл. 6.6.

Таблица 6.6. *Варианты значений переменных*

	I	II
a	4	5
b	5	6
c	4	7

Итак:

1. На рабочем листе ячейки **B2**, **B3** и **B4** отведите под переменные *a*, *b*, *c*.
2. В ячейку **B5** введите формулу нахождения искомой суммы $a + b + c$ `=СУММ(B2:B4)`.
3. В диапазон **D3:D5** введите первый вариант значений переменных *a*, *b*, *c*.
4. В диапазон **E3:E5** введите второй вариант значений переменных *a*, *b*, *c*.
5. Создайте две кнопки и, используя окно **Properties**, установите им значения свойств, как показано в табл. 6.7 (рис. 6.6).
6. В модуле рабочего листа наберите следующий код (листинг 6.3).

Таблица 6.7. *Значения свойств, установленные в окне* ***Properties***

Объект	Свойство	Значение
Кнопка	`Name`	`cmdVar1`
	`Caption`	`Вариант 1`
Кнопка	`Name`	`cmdVar1`
	`Caption`	`Вариант 2`

Нажатие кнопки **Вариант 1** произведет считывание значений при помощи свойства `Value` из ячеек диапазона **D3:D5** и ввод их с помощью свойства `Value` в ячейки диапазона **B2:B4**. Нажатие же кнопки **Вариант 2** вызовет считывание значений из ячеек диапазона **E3:E5** и ввод их в ячейки диапазона **B2:B4**. Проект кнопочного сценария готов.

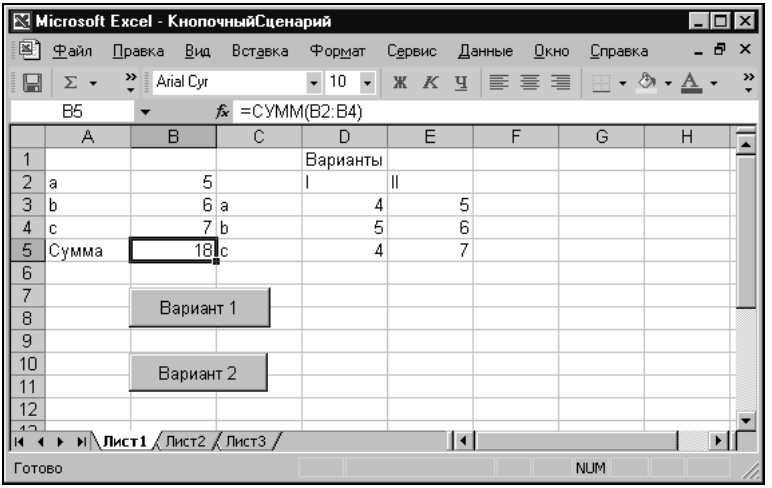


Рис. 6.6. Кнопочный сценарий

Листинг 6.3. Кнопочный сценарий. Модуль рабочего листа

```
Private Sub cmdVar1_Click()
   Range("B2").Value = Range("D3").Value
   Range("B3").Value = Range("D4").Value
   Range("B4").Value = Range("D5").Value
End Sub

Private Sub cmdVar2_Click()
   Range("B2").Value = Range("E3").Value
   Range("B3").Value = Range("E4").Value
   Range("B4").Value = Range("E5").Value
End Sub
```

Кнопочный сценарий для ввода формул с кнопками, украшенными рисунками, и пользовательским указателем мыши

Кнопки, за счет внедрения в них рисунков, могут принять более презентабельный вид. Кроме того, изменение указателя мыши, отображаемого над кнопкой, с указывающей стрелки, например, на нажимающий палец также может придать интерфейсу дополнительную визуальную привлекательность.

Рисунок загружается на поверхность кнопки при помощи свойства `Picture`. Свойство `PicturePosition` позволяет установить взаимное расположение

текста и рисунка, отображаемых на кнопке. Свойство `MousePointer` устанавливает указатель мыши. Допустимые значения этого свойства перечислены в табл. 6.8. Если значение свойства `MousePointer` равно `fmMousePointerCustom`, то курсор создается на основе CUR-файла, ссылка на который устанавливается свойством `MouseIcon`. В коде значения свойств `Picture` и `MouseIcon` задаются функцией `LoadPicture`, в качестве значения параметра которой надо указать имя соответствующего графического файла.

Таблица 6.8. *Допустимые значения свойства* `MousePointer`

Константа	Значение	Описание
`fmMousePointerDefault`	0	По умолчанию
`fmMousePointerArrow`	1	
`fmMousePointerCross`	2	
`fmMousePointerIBeam`	3	
`fmMousePointerSizeNESW`	6	
`fmMousePointerSizeNS`	7	
`fmMousePointerSizeNWSE`	8	
`fmMousePointerSizeWE`	9	
`fmMousePointerUpArrow`	10	
`fmMousePointerHourglass`	11	
`fmMousePointerNoDrop`	12	
`fmMousePointerAppStarting`	13	
`fmMousePointerHelp`	14	
`fmMousePointerSizeAll`	15	
`fmMousePointerCustom`	99	Определенный пользователем

В качестве примера использования кнопок с рисунками сконструируем еще один кнопочный сценарий, но в этот раз в ячейки рабочего листа будем вводить не числа, а формулы, по которым производятся расчеты. Абстрагируясь, рассмотрим задачу нахождения значения выражений $x + y$ и $x - y$ (рис. 6.7).

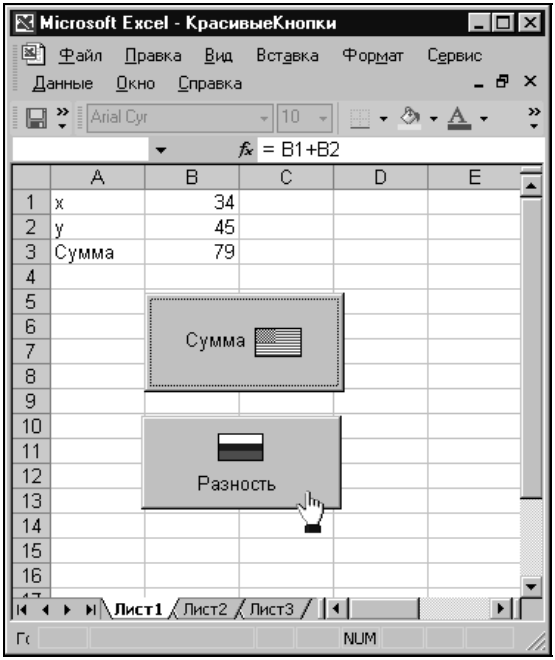


Рис. 6.7. Кнопочный сценарий для ввода формул

Итак:

1. На рабочем листе отведите ячейки **B1** и **B2** под переменные *x* и *y*. В ячейку **B3** в соответствии с выбранным сценарием будет вводиться из кода либо формула `=B1+B2`, либо `=B1-B2`.
2. Создайте две кнопки и, используя окно **Properties**, установите им значения свойств, как показано в табл. 6.9.
3. В модуле рабочего листа наберите код, показанный в листинге 6.4.

Таблица 6.9. *Значения свойств, установленные в окне* ***Properties***

Объект	Свойство	Значение
Кнопка	`Name`	`cmdSum`
	`Caption`	`Сумма`
	`Picture`	Ссылка на файл с рисунком. В данном случае была произведена ссылка на файл с изображением американского флага Flgusa02.ico
	`PicturePosition`	`fmPicturePositionRightCenter`
	`MousePointer`	`fmMousePointerAppStarting`

Таблица 6.9 (окончание)

Объект	Свойство	Значение
Кнопка	Name	cmdSub
	Caption	Разность
	Picture	Ссылка на файл с рисунком. В данном случае была произведена ссылка на файл с изображением российского флага Flgrus.ico
	PicturePosition	fmPicturePositionAboveCenter
	MousePointer	fmMousePointerCustom
	MouseIcon	Ссылка на файл с курсором. В данном случае была произведена ссылка на файл Harrow.cur, который можно найти в каталоге C:/Windows/Cursors, для создания курсора в виде указывающего пальца 

Нажатие кнопки **Сумма** произведет ввод в ячейку **B3** формулы `=B1+B2`, а в ячейку **A3** строки `Сумма`. Нажатие же кнопки **Разность** вызовет ввод в ячейку **B3** формулы `=B1-B2`, а в ячейку **A3** строки `Разность`.

Проект кнопочного сценария готов.

Листинг 6.4. Кнопочный сценарий для ввода формул

```
Private Sub cmdSum_Click()
   Range("B3").Formula = "=B1+B2"
   Range("A3").Value = "Сумма"
End Sub

Private Sub cmdSub_Click()
   Range("B3").Formula = "=B1-B2"
   Range("A3").Value = "Разность"
End Sub
```

Интерактивная кнопка и определение среднего объема продаж

Управляя событиями `MouseDown` и `MouseUp`, можно придать кнопке большую интерактивность. Продемонстрируем, как это делается, на примере простого приложения. Предположим, дорогой читатель, что вы менеджер по продажам фирмы "Русская резина", которая поставляет в страны Европы различные резинотехнические изделия. Естественно, вы постоянно ведете учет объема продаж. Для более наглядного представления данных в таблице объ-

ема продаж вы решили выделить красным цветом строку с максимальным объемом продаж и желтым — строки, в которых объем продаж выше среднего. Данные о продажах постоянно меняются, поэтому поможем вам создать такое приложение, которое бы автоматизировало процесс выделения цветом нужных строк.

Итак:

1. На рабочем листе отведите диапазон **B2:B7** под объемы продаж.
2. Введите в ячейку **B8** формулу определения суммарных продаж `=СУММ(B2:B7)`.
3. Создайте кнопку и, используя окно **Properties**, установите ей значения свойств, как показано в табл. 6.10.
4. В модуле рабочего наберите код, представленный в листинге 6.5.

Таблица 6.10. *Значения свойств, установленные в окне* ***Properties***

Объект	Свойство	Значение
Кнопка	`Name`	`cmdRefresh`
	`Caption`	`Обновить`

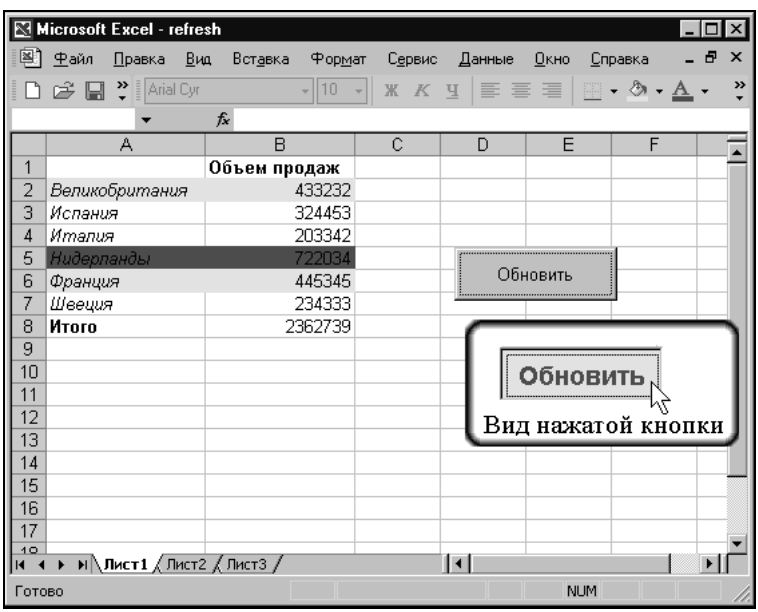


Рис. 6.8. Интерактивная кнопка и определение суммарных продаж

Нажатие кнопки **Обновить** и вызовет пересчет и переоформление таблицы (рис. 6.8). Процедуры обработки событий `MouseDown` и `MouseUp` программи-

руют различный внешний вид кнопок, а именно стиль, размер и цвет шрифта (свойства `Bold`, `Size` объекта `Font` и свойство `ForeColor` кнопки), цвет фона и вывод тени (свойства `BackColor` и `Shadow` кнопки). Изменение этих свойств кнопки и вызывает эффект дополнительной интерактивности. Пересчет таблицы производится в процедуре `DoRefresh`. Максимальное и среднее значения находятся при помощи функций рабочего листа, инкапсулированных в объект `WorksheetFunction`. Цвет ячеек устанавливается свойством `Color` объекта `Interior`. Удаление заливки ячейки реализуется установкой значения свойства `ColorIndex` равным `xlColorIndexNone`.

Листинг 6.5. Интерактивная кнопка и определение суммарных продаж

```
Private Sub cmdRefresh_MouseDown(ByVal Button As Integer, _
      ByVal Shift As Integer, ByVal X As Single, ByVal Y As Single)
   With cmdRefresh
      .Font.Bold = True
      .Font.Size = 14
      .ForeColor = vbRed
      .Shadow = True
      .BackColor = vbYellow
   End With
End Sub

Private Sub cmdRefresh_MouseUp(ByVal Button As Integer, _
      ByVal Shift As Integer, ByVal X As Single, ByVal Y As Single)
   With cmdRefresh
      .Font.Bold = False
      .Font.Size = 10
      .ForeColor = vbBlack
      .Shadow = False
      .BackColor = &H8000000F
   End With
   DoRefresh
End Sub

Sub DoRefresh()
   Dim m, a As Double
   Dim i As Integer
   With WorksheetFunction
      m = .Max(Range("B2:B7"))
      a = .Average(Range("B2:B7"))
   End With
   For i = 2 To 7
```

```
      With Range(Cells(i, 1), Cells(i, 2)).Interior
         If Cells(i, 2).Value = m Then
            .Color = RGB(255, 0, 0)
         ElseIf Cells(i, 2).Value >= a Then
            .Color = RGB(255, 255, 0)
         Else
            .ColorIndex = xlColorIndexNone
         End If
      End With
   Next
End Sub
```

Обмен значений между двумя выбранными ячейками

В качестве последнего примера по работе с кнопками на рабочем листе приведем следующий проект, способный обменивать значения между любыми двумя выделенными ячейками. Итак, на рабочем листе создайте кнопку, а в ячейки **A1**, **B1**, **A2**, **B2** введите, например, `Джеймс`, `Бонд`, `Винни`, `Пух`. В модуле рабочего листа наберите следующий код (листинг 6.6), который обрабатывает событие `Click` кнопки. Вот и все, теперь остается выбрать какие-то две из этих ячеек и нажать кнопку, а значения в них поменяются местами автоматически, при этом производится контроль за тем, чтобы выделенная область состояла ровно из двух ячеек.

Листинг 6.6. Обмен значений двух выбранных ячеек. Модуль рабочего листа

```
Private Sub CommandButton1_Click()
   Dim c1 As Range
   Dim c2 As Range
   Dim s1 As String, s2 As String

   If Selection.Cells.Count > 2 Or Selection.Cells.Count < 2 Then
      MsgBox "Выделенная область должна содержать только две ячейки", _
              vbCritical
      Exit Sub
   End If
   If Selection.Areas.Count > 1 Then
      Set c1 = Selection.Areas(1).Cells(1, 1)
      Set c2 = Selection.Areas(2).Cells(1, 1)
   ElseIf Selection.Rows.Count > Selection.Columns.Count Then
      Set c1 = Selection.Cells(1)
      Set c2 = Selection.Cells(2)
   End If
```

```
    s1 = c1.Value
    s2 = c2.Value
    c1.Value = s2
    c2.Value = s1
End Sub
```

Переключатель (*OptionButton*)

Элемент управления **Переключатель** (OptionButton) позволяет выбрать одну из нескольких взаимоисключающих альтернатив. Переключатели обычно отображаются группами по выбираемым альтернативам. Группировка производится при помощи элемента управления **Рамка** или свойства `GroupName` объекта `OptionButton`. Основными событиями переключателя являются события `Click` и `Change`. Основным свойством переключателя является свойство `Value`, возвращающее или устанавливающее его состояние. Если значение этого свойства равно `True`, то переключатель установлен, а если `False` — то сброшен.

Переключатели и объемы продаж

Для рассмотрения примера использования переключателей вернемся к отчету о продажах фирмы "Русская резина". В этот раз нам потребуется составить список тех стран, объем продаж в которые является либо максимальным, либо минимальным (рис. 6.9).

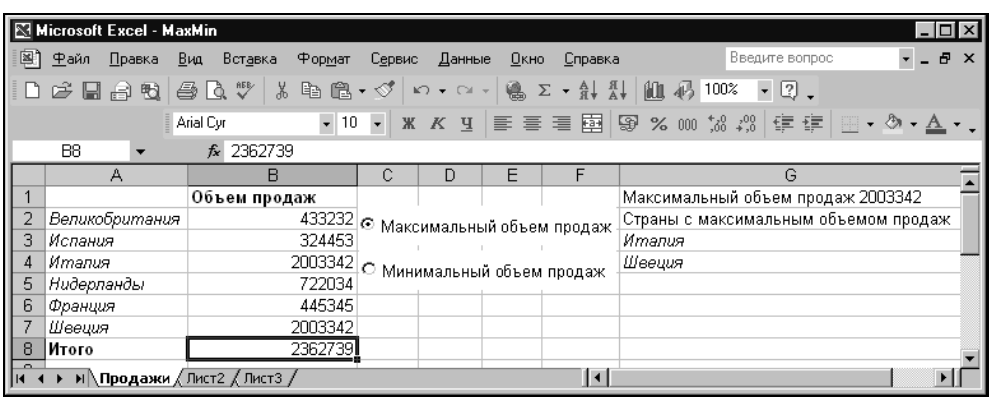


Рис. 6.9. Переключатели и объемы продаж

Итак:

1. На рабочем листе отведите диапазон **B2:B7** под объемы продаж, а диапазон **A2:A7** под названия стран, в которые эти продажи были произведены.

2. Введите в ячейку **B8** формулу определения суммарных продаж `=СУММ(B2:B7)`.
3. Создайте два переключателя и, используя окно **Properties**, установите им значения свойств, как показано в табл. 6.11.
4. В модуле рабочего листа наберите код из листинга 6.7.

Таблица 6.11. *Значения свойств, установленные в окне* ***Properties***

Объект	Свойство	Значение
Переключатель	Name	optMax
	Caption	Максимальный объем продаж
Переключатель	Name	optMin
	Caption	Минимальный объем продаж

Нажатие переключателя **Максимальный объем продаж** или **Минимальный объем продаж** приведет к определению искомого объема продаж и составлению списка соответствующих стран. Величина искомого объема продаж выводится в ячейку **G1**, пояснительная надпись — в ячейку **G2**, и далее, начиная с ячейки **G3**, выводится составленный список стран. Страны с соответствующими объемами продаж находятся методом `Find` объекта `Range`. Все найденные ячейки методом `Find` объединяются в один диапазон. На основе этого диапазона методом `Offset` объекта `Range` создается диапазон уже с названиями стран, который методом `Copy` объекта `Range` копируется в указанное место. Обратите внимание на то, что программа сама определяет диапазон, в котором надо производить поиск. Существенным является только то, что код учитывает, что первая и последняя строки столбца **B** с данными содержат не исходные данные, а заголовок поля таблицы и итоговую сумму, и поэтому их не надо учитывать, что и делается последовательным применением методов `Resize` и `Offset`. Метод `AutoFit` автоматически изменяет ширину столбца **G** с тем, чтобы в нем уместилась вся выводимая информация.

Листинг 6.7. Переключатели и объемы продаж

```
Private Sub optMax_Click()
    ListCountries True
End Sub

Private Sub optMin_Click()
    ListCountries False
End Sub
```

```
Sub ListCountries(func As Boolean)
   Dim m As Double
   Dim r As Range
   Set r = Range("b1").CurrentRegion.Columns(2)
   Set r = r.Resize(rowsize:=r.Rows.Count – 2)
   Set r = r.Offset(rowoffset:=1, columnoffset:=0)
   If func Then
      m = WorksheetFunction.Max(r)
   Else
      m = WorksheetFunction.Min(r)
   End If
   Dim c As Range
   Dim u As Range
   Dim firstAddress As String
   With r
      Set c = .Find(m, LookIn:=xlValues)
      If Not c Is Nothing Then
         firstAddress = c.Address
         Set u = c
         Do
            Set u = Application.Union(u, c)
            Set c = .FindNext(c)
       Loop While Not c Is Nothing And c.Address <> firstAddress
      End If
   End With
   Columns("G:G").Clear
   Set u = u.Offset(rowoffset:=0, columnoffset:=-1)
   If func Then
      Range("G1").Value = "Максимальный объем продаж " & m
      Range("G2").Value = "Страны с максимальным объемом продаж"
   Else
      Range("G1").Value = "Минимальный объем продаж " & m
      Range("G2").Value = "Страны с минимальным объемом продаж"
   End If
   u.Copy Range("G3")
   Columns("G:G").AutoFit
End Sub
```

Флажок (*CheckBox*) и Выключатель (*ToggleButton*)

Элемент управления **Флажок** (CheckBox) предоставляет пользователю возможность выбора. Флажок обычно имеет два состояния: установлен и сброшен, но может настраиваться на выбор из трех альтернатив. Флажок

имеет те же основные свойства `Value` и `Capture`, что и переключатель. Кроме того, флажок обладает уникальным свойством `TripleState`, позволяющим производить выбор из трех альтернатив. Допустимыми значениями свойства `TripleState` являются:

- `False` (выбор из двух альтернатив `True` и `False`, т. е. флажок может находиться только в двух состояниях — установлен и сброшен);
- `True` (выбор из трех альтернатив `True`, `False` и `Null`, т. е. флажок может находиться в трех состояниях — установлен, сброшен и нейтрален).

Элемент управления **Выключатель** (ToggleButton) предоставляет пользователю ровно те же возможности, что и флажок, но визуально он выглядит как кнопка.

Основным событием элементов управления **Флажок** и **Выключатель** является событие `Change`.

Флажок и управление отображением элементов диаграммы

Возвращаясь к отчету о продажах фирмы "Русская резина", создадим простое приложение, позволяющее управлять внешним видом диаграммы, а именно — при установленном флажке диаграмма выводится с тенью, а при снятом — без таковой (рис. 6.10).

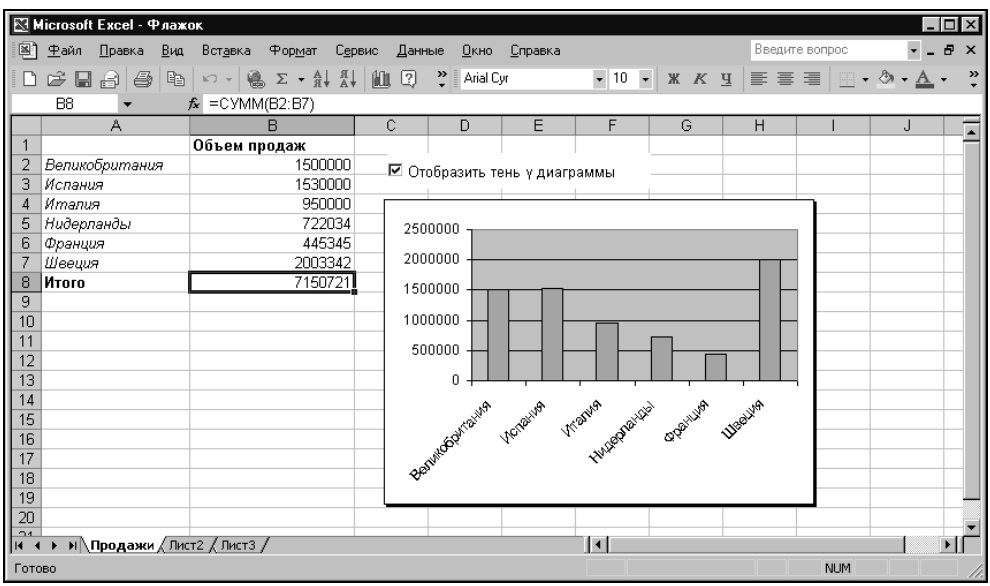


Рис. 6.10. Флажок и управление отображением элементов диаграммы

Итак:

1. На рабочем листе отведите диапазон **B2:B7** под объемы продаж, а диапазон **A2:A7** — под названия стран, в которые эти продажи были произведены.
2. На основе диапазона **A2:B7** постройте диаграмму.
3. Создайте флажок и, используя окно **Properties**, установите ему значения свойств, как показано в табл. 6.12.
4. В модуле рабочего листа наберите следующий код (листинг 6.8).

Таблица 6.12. *Значения свойств, установленные в окне* ***Properties***

Объект	Свойство	Значение
Флажок	`Name`	`chkGraph`
	`Caption`	`Отобразить тень у диаграммы`

Установка флажка **Отобразить тень у диаграммы** вызовет ее отображение с тенью, а при снятом — без таковой. Все семейство встроенных диаграмм образует семейство `ChartObjects`. Так как в нашем случае имеется единственная диаграмма, то ее можно идентифицировать по номеру, который, само собой разумеется, будет 1. Отображением тени управляет свойство `Shadow` объекта `Chart`.

Листинг 6.8. Флажок и управление отображением элементов диаграммы

```
Private Sub chkGraph_Change()
   If ActiveSheet.ChartObjects(1).Shadow Then
      ActiveSheet.ChartObjects(1).Shadow = False
   Else
      ActiveSheet.ChartObjects(1).Shadow = True
   End If
End Sub
```

Выключатель и отображение примечаний

Продемонстрируем работу выключателя на примере отчета о продажах фирмы "Русская резина". В этот проект теперь будут внедрены примечания. Установка выключателя приведет к одновременному отображению всех примечаний, а снятие выключателя — к их скрытию (рис. 6.11).

Итак:

1. На рабочем листе отведите диапазон **B2:B7** под объемы продаж, а диапазон **A2:A7** под названия стран, в которые эти продажи были произведены.

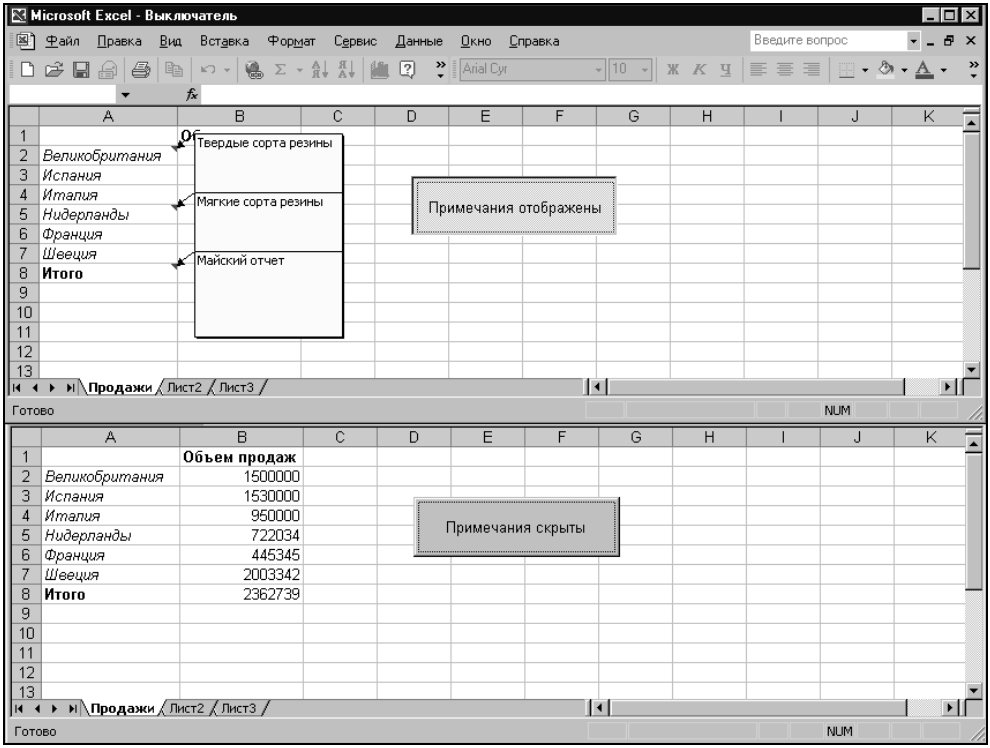


Рис. 6.11. Выключатель и отображение примечаний

2. В некоторые из ячеек диапазона **B2:B7**, используя команду **Вставка | Примечание**, введите примечания.
3. Создайте выключатели и, используя окно **Properties**, установите ему значения свойств, как показано в табл. 6.13.
4. В модуле рабочего листа наберите следующий код из листинга 6.9.

Таблица 6.13. *Значения свойств, установленные в окне* ***Properties***

Объект	Свойство	Значение
Выключатель	`Name`	`tglCom`
	`Caption`	`Примечания отображены`

Установка выключателя вызовет отображение всех примечаний, его снятие приведет к скрытию как примечаний, так и их индикаторов, и, кроме того, надпись на выключателе сменится на **Примечания скрыты**. Управление отображением примечаний и их индикаторов реализуется свойством `DisplayCommentIndicator` объекта `Application`.

Листинг 6.9. Выключатель и отображение примечаний

```
Private Sub tglCom_Click()
   If Application.DisplayCommentIndicator = xlNoIndicator Then
      Application.DisplayCommentIndicator = xlCommentAndIndicator
      tglCom.Caption = "Примечания отображены"
   Else
      Application.DisplayCommentIndicator = xlNoIndicator
      tglCom.Caption = "Примечания скрыты"
   End If
End Sub
```

Полоса прокрутки (*ScrollBar*) и Счетчик (*SpinButton*)

Элемент управления **Полоса прокрутки** (ScrollBar) применяется для установки числового значения, причем этот элемент может устанавливать только целые неотрицательные значения. Основными событиями элемента управления **Полоса прокрутки** являются `Change`, `SpinUp` и `SpinDown`. В табл. 6.14 перечислены наиболее часто используемые свойства полосы прокрутки.

Таблица 6.14. *Свойства полосы прокрутки*

Свойство	Описание
`Value`	Возвращает или устанавливает текущее значение полосы прокрутки, которое может быть только целочисленным
`Min`	Минимальное значение полосы прокрутки
`Max`	Максимальное значение полосы прокрутки
`SmallChange`	Устанавливает шаг изменения значения при щелчке на одной из стрелок полосы прокрутки
`LargeChange`	Устанавливает шаг изменения значения при щелчке между ползунком и одной из стрелок полосы прокрутки
`LinkedCell`	Ссылка на ячейку значения свойства `Value`, синхронизированного со значением аналогичного свойства полосы прокрутки
`Orientation`	Устанавливает ориентацию полосы прокрутки. Допустимыми значениями являются следующие константы: `fmOrientationAuto` (ориентация зависит от размера элемента управления. Используется по умолчанию), `fmOrientationVertical` (вертикальное расположение), `fmOrientationHorizontal` (горизонтальное расположение)

Элемент управления **Счетчик** (SpinButton) по своим функциональным возможностям аналогичен полосе прокрутки. Если не быть чрезмерным буквоедом, то можно сказать, что счетчик — это полоса прокрутки без ползунка. Счетчик имеет те же свойства `Value`, `Min`, `Max` и `SmallChange`, что и полоса прокрутки.

Ввод значений в ячейку и управление цветом

В качестве примера использования полос прокрутки создадим приложение, демонстрирующее RGB-цветовую модель. Итак, на рабочем листе расположите три полосы прокрутки (рис. 6.12). Используя окно **Properties**, установите им значения свойств, как показано в табл. 6.15.

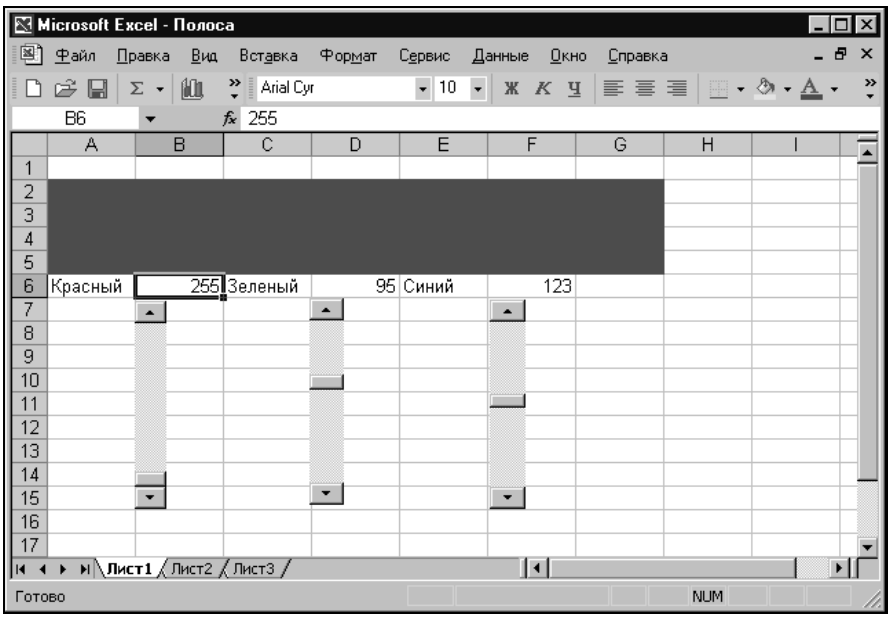


Рис. 6.12. Ввод значений в ячейку и управление цветом

Таблица 6.15. *Значения свойств, установленные в окне* ***Properties***

Объект	Свойство	Значение
Полоса прокрутки	`Name`	`scrRed`
	`Min`	`0`
	`Max`	`255`
	`LinkedCell`	`B6`

Таблица 6.15 *(окончание)*

Объект	Свойство	Значение
Полоса прокрутки	Name	scrGreen
	Min	0
	Max	255
	LinkedCell	D6
Полоса прокрутки	Name	scrBlue
	Min	0
	Max	255
	LinkedCell	F6

В модуле рабочего листа наберите следующий код (листинг 6.10). Каждая из полос прокрутки может изменять свое значение в пределах от 0 до 255, причем текущее значение отображается в той ячейке, на которую приведена ссылка в свойстве `LinkedCell`. Более того, эти ячейки работают согласованно с полосами прокрутки. Если пользователь изменит значение в ячейке, то одновременно изменится и значение свойства `Value` полосы прокрутки. Каждая из полос прокрутки отвечает за один из трех компонентов (красного, зеленого и синего) RGB-цветовой модели, в соответствии с которой устанавливается цвет фона диапазона **A2:G5**.

Листинг 6.10. Выключатель и отображение примечаний

```
Private Sub scrRed_Change()
   SetColor scrRed.Value, scrGreen.Value, scrBlue.Value
End Sub

Private Sub scrGreen_Change()
   SetColor scrRed.Value, scrGreen.Value, scrBlue.Value
End Sub

Private Sub scrBlue_Change()
   SetColor scrRed.Value, scrGreen.Value, scrBlue.Value
End Sub

Private Sub SetColor(r As Integer, g As Integer, b As Integer)
   Range("A2:G5").Interior.Color = RGB(r, g, b)
End Sub
```

Ввод в ячейку с помощью полосы прокрутки и счетчика нецелочисленных значений

Свойство `Value` полосы прокрутки и счетчика может принимать только целочисленные значения. Если же надо управлять нецелочисленными значениями, то их просто следует масштабировать. Как это делается, покажем на следующем демонстрационном проекте, в котором построен график функции $F(x) = \cos(ax) \times \sin(bx)$. Значения же параметров a и b лежат в диапазоне от 0 до 10 и могут изменяться с шагом 0,1. Данные значения вводятся в ячейки с помощью счетчиков, а график при этом автоматически перестраивается (рис. 6.13).

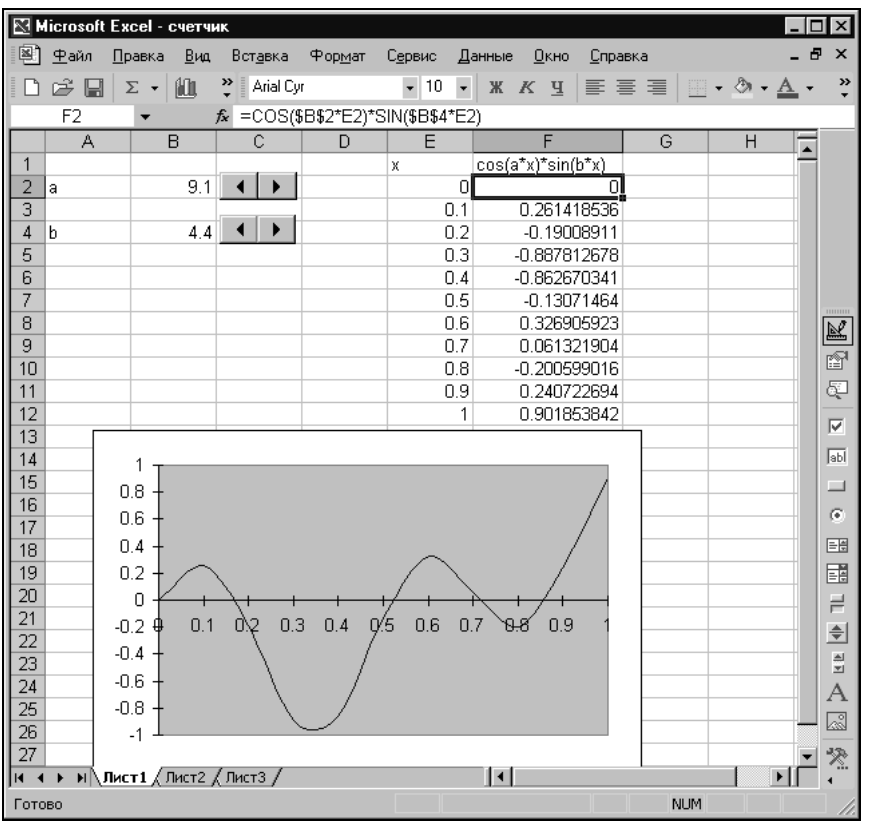


Рис. 6.13. Ввод в ячейку счетчиком нецелочисленных значений

Итак:

1. На рабочем листе отведите под значения параметров a и b ячейки **B2** и **B4**.

2. В диапазон ячеек **E2:E12** введите результат табулирования переменной *x* от значения 0 до 1 с шагом 0,1.
3. В ячейку **F2** введите формулу `=COS($B$2*E2)*SIN($B$4*E2)`.
4. Выберите ячейку **F2**, расположите указатель мыши на маркере заполнения и протяните его вниз на диапазон **F2:F12**.
5. На основе диапазона **E2:F12** с помощью мастера диаграмм постройте график.
6. Создайте два счетчика и, используя окно **Properties**, установите им значения свойств, как показано в табл. 6.16.
7. В модуле рабочего листа наберите следующий код (листинг 6.11).

Таблица 6.16. *Значения свойств, установленные в окне* ***Properties***

Объект	Свойство	Значение
Счетчик	Name	spnA
	Min	0
	Max	100
Счетчик	Name	spnB
	Min	0
	Max	100

Процедуры обработки события `Change` обеспечивают ввод масштабированных значений счетчиков в ячейки рабочего листа. Процедура же обработки события `Change` рабочего листа применяется для синхронизации работы ячеек и счетчиков, а именно изменение значений в ячейках **B2** и **B4** приводит к изменению значений счетчиков.

Листинг 6.11. Ввод в ячейку нецелочисленных значений

```
Private Sub spnA_Change()
   Range("B2").Value = spnA.Value / 10
End Sub

Private Sub spnB_Change()
   Range("B4").Value = spnB.Value / 10
End Sub

Private Sub Worksheet_Change(ByVal Target As Range)
   If Target.Address = "$B$2" Then
      spnA.Value = CInt(Range("B2").Value * 10)
```

```
   ElseIf Target.Address = "$B$4" Then
      spnB.Value = CInt(Range("B4").Value * 10)
   End If
End Sub
```

Список (*ListBox*)

Элемент управления **Список** (ListBox) применяется для хранения списка значений. В списке пользователь может выбрать одно или несколько значений, которые в последующем используются в тексте программы. Обратите внимание, что на этапе конструирования визуально список похож на поле ввода. Обычно выбор элемента из списка производится щелчком на элементе. Двойной же щелчок на элементе применяется для выполнения каких-то действий в программе, связанных с этим элементом. Событие `Change` генерируется при смене выбранного элемента.

В табл. 6.17 и 6.18 приведены наиболее часто используемые свойства и методы списка.

Таблица 6.17. *Свойства списка*

Свойство	Описание
`ListIndex`	Возвращает номер выбранного элемента списка. Нумерация элементов списка начинается с нуля. Если ни один элемент списка не выбран, то возвращает –1
`ListCount`	Возвращает число элементов списка
`TopIndex`	Возвращает элемент списка с наибольшим номером
`ColumnCount`	Устанавливает число столбцов в списке
`TextColumn`	Устанавливает столбец в списке, элементы из которого возвращаются в качестве значения свойства `Text`
`Text`	Возвращает выбранный в списке элемент
`List`	Возвращает элемент списка, стоящий на пересечении указанной строки и столбца
`ListFillRange`	Ссылка на диапазон, из которого заполняется список
`RowSource`	Устанавливает диапазон, содержащий элементы списка
`ControlSource`	Устанавливает диапазон (ячейку), куда возвращается выбранный элемент из списка

Таблица 6.17 *(окончание)*

Свойство	Описание
MultiSelect	Устанавливает способ выбора элементов списка. Допустимые значения: fmMultiSelectSingle (выбор только одного элемента), fmMultiSelectMulti (разрешен выбор нескольких элементов, выбор осуществляется либо щелчком, либо нажатием клавиши <Пробел>), fmMultiSelectExtended (разрешено использование клавиши <Shift> при выборе ряда последовательных элементов списка)
Selected	Логическое свойство, которое возвращает значение True, если элемент списка выбран, и False — в противном случае. Используется для определения выбранного элемента, когда значение свойства MultiSelect установлено равным fmMultiSelectMulti или fmMultiSelectExtended
ColumnWidths	Устанавливает ширину столбцов списка
ColumnHeads	Логическое свойство, определяющее, выводить ли в списке заголовки столбцов
ListStyle	Устанавливает способ выделения выбранных элементов. Допустимые значения: fmListStylePlain (выбранный элемент из списка выделяется цветом), fmListStyleOption (перед каждым элементом в списке располагается флажок, и выбор элемента из списка соответствует установке этого флажка).
MatchEntry	Выводит первый подходящий элемент из списка при наборе его имени с клавиатуры. Допустимые значения: fmMatchEntryNone (режим вывода подходящего элемента в списке отключен), fmMatchEntryFirstLetter (выводит подходящий элемент по набранной первой букве. В этом случае, предпочтительно, чтобы элементы списка были упорядочены в алфавитном порядке), fmMatchEntryComplete (выводит подходящий элемент по полному набранному имени)
BoundColumn	Устанавливает данные, возвращаемые свойством Value. Допустимые значения: 0 (свойством Value возвращается индекс выбранной строки, т. е. в этом случае оно действует как свойство ListIndex), от 1 до количества столбцов в списке (свойством Value возвращается элемент из выбранной строки, стоящий в столбце, заданном значением свойства BoundColumn)

Таблица 6.18. *Методы списка*

Метод	Описание
Clear	Удаляет все элементы из списка
RemoveItem	Удаляет элемент из списка с указанным значением индекса
AddItem	Добавляет элемент в список

Сценарии со списком

В качестве примера использования списков вернемся к сценарию нахождения суммы $a + b + c$ при различных вариантах значений слагаемых. Только в этот раз управлять сценариями будем не кнопками, а списком (рис. 6.14).

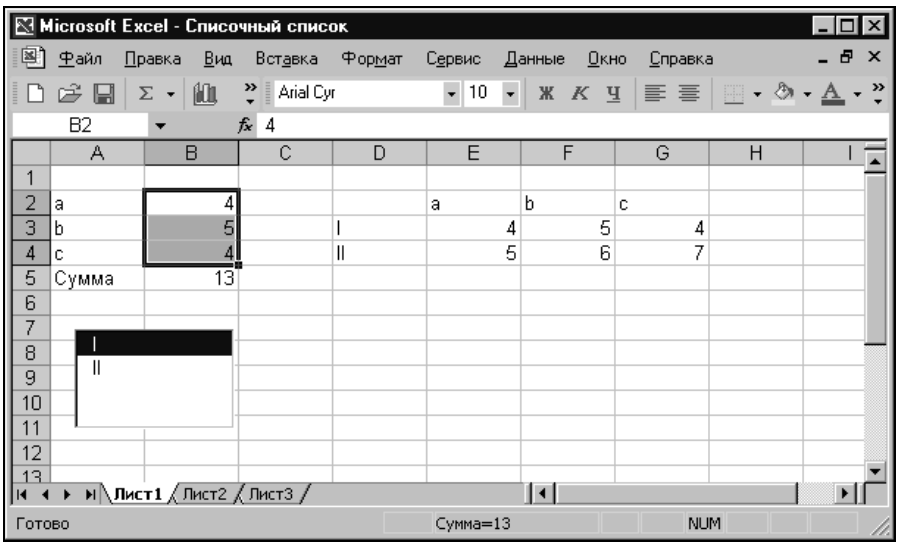


Рис. 6.14. Сценарии со списком

Итак:

1. На рабочем листе ячейки **B2**, **B3** и **B4** отведите под значения слагаемых *a*, *b* и *c*.
2. В ячейку **B5** введите формулу `=СУММ(B2:B4)`.
3. В диапазон **E3:G3** введите первый вариант возможных значений слагаемых *a*, *b* и *c*, а в ячейку **D3** — название первого варианта.
4. В диапазон **E4:G4** введите второй вариант возможных значений слагаемых *a*, *b* и *c*, а в ячейку **D4** — название второго варианта.
5. Создайте список и, используя окно **Properties**, установите ему значения свойств, как показано в табл. 6.19.

Таблица 6.19. *Значения свойств, установленные в окне* ***Properties***

Объект	Свойство	Значение
Список	`Name`	`lstVar`
	`ListFillRange`	`D3:D4`

В модуле рабочего листа наберите следующий код (листинг 6.12). Свойство `ListFillRange` заполняет список именами вариантов на основе тех значений, которые введены в указанный диапазон. Процедура обработки события `Change` списка при изменении выбранного элемента определяет его индекс, который возвращается свойством `ListIndex`. Этот индекс позволяет также идентифицировать диапазон, в котором хранятся значения слагаемых выбранного варианта. Специфицированный диапазон сначала копируется в буфер обмена, а затем с помощью специальной вставки с транспонированием вставляется в диапазон **B2:B4**, отведенный под слагаемые.

Листинг 6.12. Сценарии со списком

```
Private Sub lstVar_Change()
   Dim i As Integer
   i = lstVar.ListIndex
   Range(Cells(3 + i, 5), Cells(3 + i, 7)).Copy
   Range("B2").PasteSpecial Paste:=xlPasteAll, Transpose:=True
End Sub
```

Защита ячеек рабочего листа

В проектах с пользовательским интерфейсом часто необходима поддержка целостности, чтобы пользователь не мог нарушать его структуру и производил только позволенные в соответствии с бизнес-сценарием действия, а именно, чтобы ему был разрешен ввод данных только в специфицированные ячейки, а попытки ввода в остальные ячейки — блокировалась программно. Подобный эффект достигается использованием метода `Protect`, устанавливающего защиту на рабочий лист со значением параметра `UserInterfaceOnly`, равным `True`. Эта установка запрещает ввод пользователем данных во все незаблокированные ячейки, в то же время позволяя это делать программно. Свойство же `Locked` контролирует блокировку ячеек.

Следующий простой проект (листинг 6.13) демонстрирует работу с методом `Protect` и свойством `Locked`. Итак, по сценарию задачи требуется найти либо сумму, либо разность двух чисел, введенных в ячейки **B1** и **B2**. Выбор операции пользователь производит из списка. Ему также разрешен ввод чисел в ячейки **B1** и **B2**, а все остальные ячейки для него заблокированы. Ввод же названия операции и формулы в ячейки **A3** и **B3** реализуется программно по сделанному пользователем в списке выбору (рис. 6.15).

Листинг 6.13, *а*. Защита ячеек рабочего листа. Модуль `ЭтаКнига`

```
Private Sub Workbook_Open()
  With Worksheets(1).ListBox1
      .Clear
```

```
      .AddItem "Сумма"
      .AddItem "Разность"
   End With
   On Error Resume Next
   Worksheets(1).Range("B1:B2").Locked = False
   Worksheets(1).Protect Password:="1", UserInterfaceOnly:=True
End Sub
```

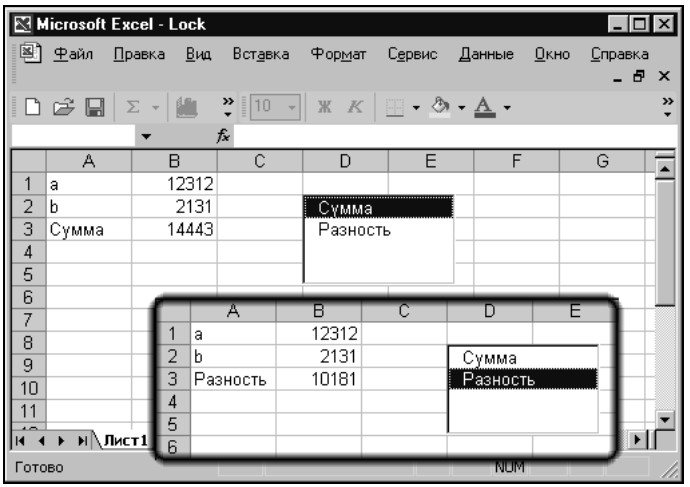


Рис. 6.15. Защита ячеек рабочего листа

Листинг 6.13, *б*. Защита ячеек рабочего листа. Модуль рабочего листа

```
Private Sub ListBox1_Click()
   Select Case ListBox1.ListIndex
      Case 0
         Range("A3").Value = "Сумма"
         Range("B3").Formula = "=B1+B2"
      Case 1
         Range("A3").Value = "Разность"
         Range("B3").Formula = "=B1-B2"
   End Select
End Sub
```

Управление печатью элементов управления

В выводимом на печать отчете иногда требуется отображать элементы управления, а иногда не требуется, если они выполняют вспомогательную роль. В этом случае на помощь приходит свойство `PrintObject` элементов

управления, которое устанавливает, надо ли выводить элемент управления при печати. Если значение этого свойства равно `True`, то объект печатается, а если `False`, то не печатается. Для демонстрации работы этого свойства модифицируем проект из предыдущего раздела. Добавьте на рабочий лист две кнопки и флажок. Если флажок **Печатать элементы управления** установлен, то эти элементы и список выводятся на печать, а если снят, то не выводятся. Кнопка **Печать** отвечает за печать, а кнопка **Предварительный просмотр** — за предварительный просмотр рабочего листа перед печатью. На рис. 6.16 показан результат просмотра рабочего листа при условии, что флажок **Предварительный просмотр** снят.

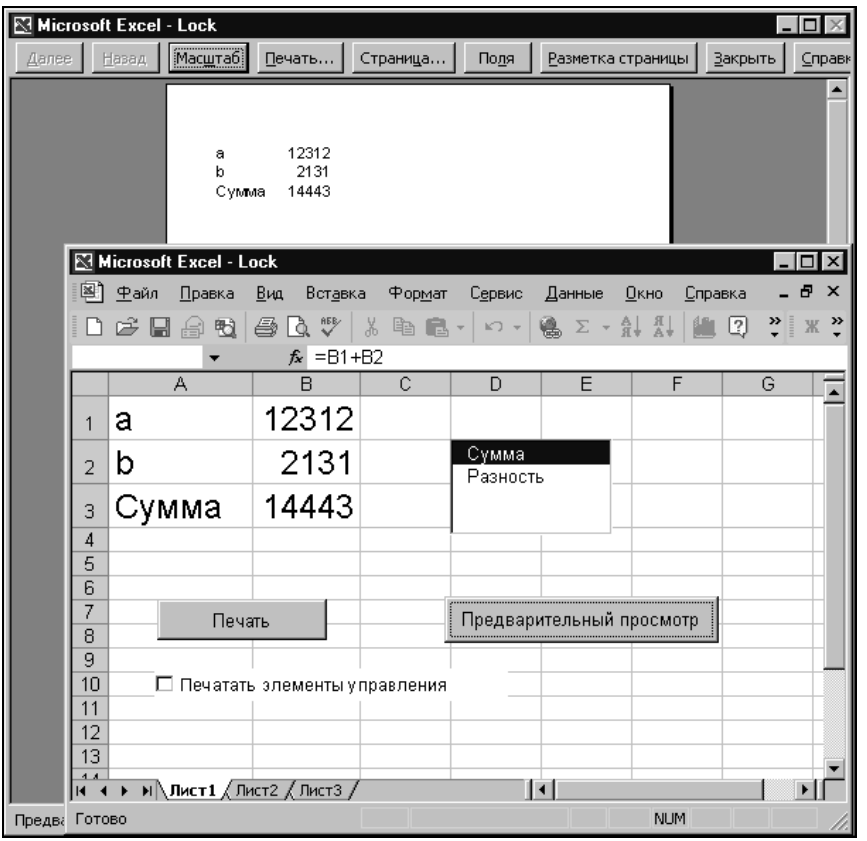


Рис. 6.16. Управление печатью элементов управления

Создайте список и, используя окно **Properties**, установите ему значения свойств, как показано в табл. 6.19.

В модуль рабочего листа добавьте код, представленный в листинге 6.14. Проект готов.

Таблица 6.19. *Значения свойств, установленные в окне* ***Properties***

Объект	Свойство	Значение
Кнопка	Name	cmdPrint
	Caption	Печать
Кнопка	Name	cmdPreview
	Caption	Предварительный просмотр
Флажок	Name	chkPrint
	Caption	Печатать элементы управления

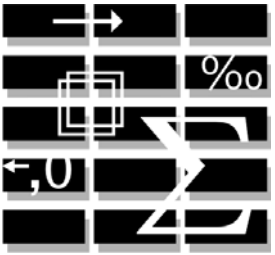
Листинг 6.14. Печать элементов управления

```
Private Sub cmdPreview_Click()
   SetPrintStatus
   ActiveSheet.PrintPreview
End Sub

Private Sub cmdPrint_Click()
   SetPrintStatus
   ActiveSheet.PrintOut
End Sub

Private Sub SetPrintStatus()
   If chkPrint.Value Then
      ListBox1.PrintObject = True
      cmdPrint.PrintObject = True
      cmdPreview.PrintObject = True
      chkPrint.PrintObject = True
   Else
      ListBox1.PrintObject = False
      cmdPrint.PrintObject = False
      cmdPreview.PrintObject = False
      chkPrint.PrintObject = False
   End If
End Sub
```

ГЛАВА 7



Форма и элементы управления

Форма

По своей сути форма (или пользовательская форма) представляет собой диалоговое окно, в котором можно размещать различные элементы управления. В приложении может быть как одна, так и несколько форм. Используя комбинацию форм и элементов управления, возможно создать любой пользовательский интерфейс, максимально отображающий бизнес-логику проекта и предоставляющий оптимальные удобства пользователю.

Добавление формы в проект

Диалоговое окно является формой. Чтобы добавить форму в проект:

1. Перейдите в редактор Visual Basic.
2. Выберите команду **Insert | UserForm**.

Новая форма добавлена в проект (рис. 7.1).

Примечание

Размеры формы можно изменять при помощи маркеров изменения размеров.

Семейство форм

Семейство `UserForms` является семейством, компоненты которого представляют все загруженные формы в приложении. Как и у всех семейств, у семейства `UserForms` имеются свойства `Count` (возвращает число компонентов в семействе) и `Item` (возвращает определенный компонент семейства), а также метод `Add` (добавляет к семейству новый компонент).

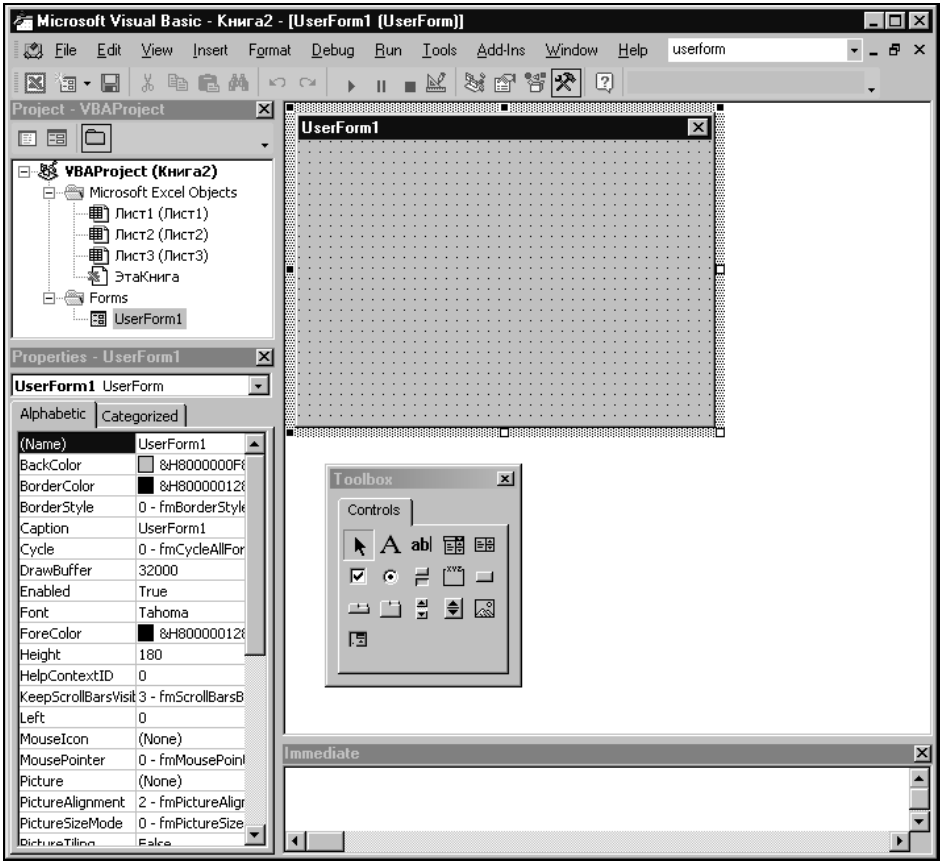


Рис. 7.1. Новая форма в редакторе Visual Basic

Свойства формы

Форма обладает широким спектром свойств, позволяющих контролировать как ее внешний вид, так и параметры функционирования. Конечно, наиболее часто используемыми свойствами формы являются те, которые задают имя формы и текст, отображаемый в заголовке окна. В табл. 7.1 перечислены свойства формы.

Таблица 7.1. *Свойства формы*

Свойство	Описание
`Name`	Имя формы
`ActiveControl`	Возвращает ссылку на элемент управления, получивший фокус

Таблица 7.1 *(продолжение)*

Свойство	Описание
BackColor	Цвет фона
BorderColor	Цвет границы
BorderStyle	Стиль границы. Допустимыми значениями являются следующие константы: fmBorderStyleNone и fmBorderStyleSingle
CanPaste	Устанавливает, возможна ли вставка объекта из буфера обмена
CanRedo	Устанавливает, возможна ли операция **Повторить**
CanUndo	Устанавливает, возможна ли операция **Отменить**
Caption	Заголовок формы
Cycle	Специфицирует действия элементов, расположенных в объектах-контейнерах **Frame** и **Page**, выполняемые при потере фокуса
DrawBuffer	Задает размер памяти, используемой при перерисовке изображения
Enabled	Определяет, достижима ли для пользователя форма
ForeColor	Цвет заголовка
Height, Width	Высота и ширина формы
HelpContextID	Ссылка на главу справочного файла
InsideHeight, InsideWidth	Высота и ширина пользовательской части формы, т. е. без строки заголовка и толщины границы
KeepScrollBarsVisible	Отображение полос прокрутки. Допустимыми значениями являются следующие константы: fmScrollBarsNone, fmScrollBarsHorizontal, fmScrollBarsVertical и fmScrollBarsBoth
Left, Top	Координаты левого верхнего угла формы
MouseIcon	Назначает пользовательский указатель мыши
MousePointer	Специфицирует тип указателя мыши
Picture	Задает ссылку на файл с растровым изображением, используемым в качестве фона
PictureAlignment	Специфицирует выравнивание растрового изображения, используемого в качестве фона
PictureSizeMode	Определяет, надо ли изменять масштаб изображения

Таблица 7.1 *(окончание)*

Свойство	Описание
`ScrollHeight`, `ScrollWidth`	Задают высоту и ширину прокручиваемой области
`ScrollLeft`, `ScrollTop`	Устанавливают координату верхнего левого угла прокручиваемой области
`SpecialEffect`	Определяет внешний вид формы
`StartUpPosition`	Специфицирует начальное местоположение формы
`Tag`	Задает параметр, используемый для идентификации конкретной формы
`VerticalScrollbarSide`	Определяет, на какой стороне формы отображаются полосы прокрутки
`Visible`	Устанавливает видимость формы
`WhatsThisButton`	Специфицирует отображение кнопки с вопросительным знаком
`Zoom`	Указывает, на сколько надо изменить размер отображаемого объекта

Методы формы

Форма имеет множество методов, позволяющих реализовывать широкий спектр операций от ее отображения или скрытия до перерисовки изображения. В табл. 7.2 перечислены методы формы.

Таблица 7.2. *Методы формы*

Метод	Описание
`Copy`	Копирует содержание объекта в буфер обмена
`Cut`	Копирует с удалением содержание объекта в буфер обмена
`Hide`	Скрывает форму без удаления ее из памяти
`Load`	Загружает объект в память без его отображения
`Move`	Перемещает форму
`Paste`	Вставляет содержимое буфера обмена
`PrintForm`	Печатает изображение формы
`RedoAction`	Повторяет последнюю команду **Повторить**

Таблица 7.2 *(окончание)*

Метод	Описание
`Repaint`	Обновляет изображение формы
`Scroll`	Прокручивает изображение
`SetDefaultTabOrder`	Устанавливает используемый по умолчанию порядок обхода элементов управления клавишей <Tab>
`Show`	Отображает форму
`UndoAction`	Повторяет последнюю команду **Отменить**
`Unload`	Удаляет объект из памяти
`WhatsThisMode`	Отображает указатель с вопросительным знаком

События формы

Процедуры обработки событий позволяют создавать программы, дающие возможность контролировать весь жизненный цикл формы от ее инициализации до закрытия. В табл. 7.3 перечислены события формы.

Таблица 7.3. *События формы*

Событие	Описание
`Activate`, `Deactivate`	Происходят при активизации и деактивизации формы
`AddControl`	Происходит при добавлении элемента управления
`BeforeDragOver`	Происходит при буксировке данных
`BeforeDropOrPaste`	Происходит перед вставкой данных, производимой буксировкой
`Click`	Происходит, когда пользователь щелкает мышью на форме
`DblClick`	Происходит, когда пользователь дважды щелкает мышью на форме
`Error`	Происходит, когда форма обнаружила ошибку, но не может передать сообщение
`Initialize`	Происходит при инициализации формы
`Layout`	Происходит при изменении местоположения формы
`KeyDown`, `KeyUp`	Происходят, когда пользователь нажимает и отпускает любую клавишу на клавиатуре, а форма имеет фокус

Таблица 7.3 (окончание)

Событие	Описание
KeyPress	Происходит, когда пользователь нажимает любую клавишу на клавиатуре, кроме функциональных клавиш, клавиш управлением курсором и клавиш управления, а форма имеет фокус
MouseDown, MouseUp	Происходят, когда пользователь нажимает и отпускает любую кнопку мыши
MouseMove	Происходит, когда пользователь передвигает указатель мыши над формой
QueryClose	Происходит перед закрытием окна
RemoveControl	Происходит при удалении элемента управления
Resize	Происходит при изменении размеров элемента управления
Scroll	Происходит при прокрутке
Terminate	Происходит при закрытии формы
Zoom	Происходит при изменении масштабирования формы

Отображение и скрытие формы

При работе с формами особое место занимают метод и два оператора, которые управляют процессами начала и завершения работы с формой:

- метод Show загружает форму в память и отображает ее;
- оператор Unload выгружает форму с экрана и из памяти;
- оператор End завершает выполнение кода без генерации событий Unload или Terminate. Поэтому завершение работы приложения по инструкции End игнорирует код, написанный в процедурах, обрабатывающих перечисленные события.

Отображать и скрывать форму можно с помощью методов Show и Hide.

Первый проект с формой

В качестве первого проекта с формой создадим пользовательскую форму, которая отображается на экране при нажатии кнопки, расположенной на рабочем листе (рис. 7.2).

Итак, создайте форму и, используя окно **Properties**, установите ей значения свойств, как показано в табл. 7.4.

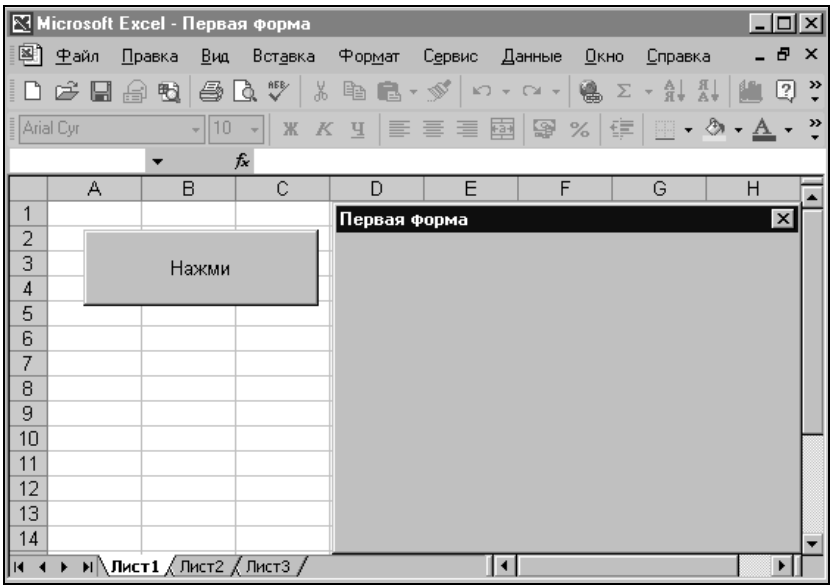


Рис. 7.2. Первый проект с формой

Таблица 7.4. *Значения свойств формы, установленные в окне* ***Properties***

Объект	Свойство	Значение
Форма	`Name`	`frmFirst`
	`Caption`	`Первая форма`

Перейдите на рабочий лист, создайте кнопку и, используя окно **Properties**, установите ей значения свойств, как показано в табл. 7.5.

Таблица 7.5. *Значения свойств кнопки, установленные в окне* ***Properties***

Объект	Свойство	Значение
Кнопка	`Name`	`cmdDemoForm`
	`Caption`	`Нажми`

В модуле рабочего листа наберите следующий код (листинг 7.1). Теперь при нажатии кнопки на экране отобразится форма.

Листинг 7.1. Первая форма. Модуль рабочего листа

```
Private Sub cmdDemoForm_Click()
   frmFirst.Show
End Sub
```

Как запустить проект на исполнение

Форму можно связать с любым элементом управления, размещенным на рабочем листе. Как это делается, было показано в предыдущем примере. В последующих примерах, где основными объектами являются сама форма и функциональные возможности размещенных на ней элементов управления, мы не будем повторять то, как форма интегрируется в рабочий лист, оставляя составление соответствующего кода читателю.

Для проверки работы кода, связанного с формой, на самом деле нет необходимости создавать элементы управления на рабочем листе и с ними связывать форму. Достаточно после конструирования формы и написания кода в модуле формы выбрать команду **Run | Run Sub/UserForm**, либо нажать клавишу <F5>, либо кнопку  панели инструментов **Standard**, и форма отобразится поверх активного рабочего листа.

Ключевое слово *Me*

В коде часто используется ключевое слово `Me`, которое возвращает имя активного окна. Например, вместо кода

```
UserForm1.Caption = "Пример"
Unload UserForm1
```

обычно используют следующий код:

```
Me.Caption = "Пример"
Unload Me
```

Форма с обновляемым фоновым рисунком

Рисунок в качестве фона можно внедрить в форму при помощи свойства `Picture`. Это свойство отображает рисунок в его оригинальных размерах. Если же требуется, чтобы рисунок занимал либо всю клиентскую часть формы, либо всю ее ширину или высоту, надо воспользоваться свойством `PictureSizeMode`. Свойство `PictureAlignment` размещает рисунок в клиентской области формы, например, по ее центру или прижатым к специфицированным сторонам формы.

Сконструируем проект формы, в которой в качестве фона отображается рисунок. При щелчках же на форме данный и еще один рисунок будут поочередно заменять друг друга.

Для реализации проекта необходимо иметь два растровых рисунка. В данном случае это C:\View.jpg и C:\Crocodiles.jpg. Итак, сконструируйте форму и, используя окно **Properties**, установите ей значения свойств, как показано в табл. 7.6.

Таблица 7.6. *Значения свойств, установленные в окне* ***Properties***

Объект	Свойство	Значение
Форма	`Picture`	Ссылка на растровый файл C:\Crocodiles.jpg
	`PictureSizeMode`	`fmPictureSizeModeStretch`
	`Caption`	`Фоновые сменяемые рисунки`

В модуле формы наберите следующий код (листинг 7.2). Теперь при щелчке рисунки C:\View.jpg и C:\Crocodiles.jpg будут на форме поочередно друг друга заменять. Загрузка изображения из файла в коде производится функцией `LoadPicture`, в качестве значения параметра которой указывается имя исходного файла. Так как значение свойства `PictureSizeMode` для рисунка Crocodiles.jpg установлено равным `fmPictureSizeModeStretch`, то он растягивается или сжимается с нарушением пропорций с тем, чтобы занять всю клиентскую область формы (рис. 7.3, *а*). Значение свойства `PictureSizeMode` для рисунка C:\View.jpg установлено равным `fmPictureSizeModeZoom`, поэтому он растягивается или сжимается с сохранением пропорций с тем, чтобы занять либо всю ширину, либо высоту клиентской области формы (рис. 7.3, *б*). Свойство `PictureAlignment`, установленное равным `fmPictureAlignmentTopLeft`, приводит к тому, что левые верхние углы рисунка и клиентской области формы совпадают.



а *б*

Рис. 7.3. Форма с обновляемым фоновым рисунком

Листинг 7.2. Форма с обновляемым фоновым рисунком

```
Private Sub UserForm_Click()
   Static flag As Boolean
   Dim filename As String
   If Not flag Then
      filename = "C:\Crocodiles.jpg"
      Me.Picture = LoadPicture(filename)
      Me.PictureSizeMode = fmPictureSizeModeStretch
      Me.Caption = "Фоновые сменяемые рисунки " & filename
   Else
      filename = "C:\View.jpg"
      Me.Picture = LoadPicture(filename)
      Me.PictureSizeMode = fmPictureSizeModeZoom
      Me.PictureAlignment = fmPictureAlignmentTopLeft
      Me.Caption = "Фоновые сменяемые рисунки " & filename
   End If
   Me.Repaint
   flag = Not flag
End Sub
```

Примечание

Для загрузки рисунка совсем не обязательно было устанавливать значения свойств `Picture` и `PictureSizeMode` формы. Достаточно было в модуль формы добавить следующий код, который на этапе инициализации формы вызывает процедуру обработки события `Click` формы, в которой и происходит отображение

```
Private Sub UserForm_Initialize()
   UserForm_Click
End Sub
```

Удаление рисунка

В окне **Properties** картинка удаляется размещением курсора в поле **Picture** и нажатием клавиши <Delete>. В коде это достигается присвоением значения свойства `Picture` равным `LoadPicture("")`. Например:

```
Me.Picture = LoadPicture("")
```

Форма с мозаичным фоном и установкой свойств на этапе инициализации

Изображение в форму можно выводить не только в виде целой картинки, но и как мозаику. В этом случае значение свойства `PictureTiling` надо

установить равным `True`. Конечно, следует позаботиться и о свойстве `PictureAlignment`, которое задает расположение первоначальной картинки, от которой все мозаичное покрытие и строится (рис. 7.4).

Значения свойств формы можно устанавливать как при помощи окна **Properties**, так и в коде. В последнем случае это, как правило, делается в процедуре обработке события `Initialize` формы, которое генерируется при инициализации формы, но до ее отображения на экране.

В качестве примера построим форму с мозаичным фоном и заданием ее свойств в коде на этапе инициализации формы. Итак, создайте форму и в модуле формы наберите следующий код (листинг 7.3). Кроме того, убедитесь, что в каталоге, в котором сохранена рабочая книга, находится Me.bmp или любой другой файл, который вы хотите отобразить в виде мозаичного фона.

Данное приложение проверяет наличие файла с растровым изображением в данном каталоге при помощи функции `Dir`. Если его нет в рабочем каталоге, то при запуске приложения форма будет отображаться без мозаичного фона. Функция `Dir` возвращает имя каталога или файла, соответствующего образцу, указанному в виде аргумента этой функции. В образце можно использовать символы "*" и "?". Если подходящего каталога или файла нет, то функция `Dir` возвращает пустую строку. Поэтому проверка наличия файла просто сводится к проверке длины возвращаемой функцией `Dir` строки. Если она имеет нулевую длину, то файла нет.



Рис. 7.4. Форма с мозаичным фоном

Листинг 7.3. Мозаика

```
Private Sub UserForm_Initialize()
   Me.Caption = "Мозаика"
   Me.BorderStyle = fmBorderStyleNone
   Dim pic As String
   pic = CurDir & "\" & "me.bmp"
   If Len(Dir(pic)) > 0 Then
      Me.Picture = LoadPicture(pic)
      Me.PictureAlignment = fmPictureAlignmentTopLeft
      Me.PictureTiling = True
   Else
      MsgBox "Нет файла " & CurDir & "\" & "me.bmp"
   End If
End Sub
```

Закрытие формы при нажатии клавиши <Esc>

Нажатие кнопки **Закрыть**, расположенной в правом верхнем углу формы, приводит к ее закрытию. Возникает естественный вопрос: "Возможно ли закрытие формы при нажатии каких-либо клавиш клавиатуры, например <Esc>?" Ответ, конечно, положителен. Для этого надо всего лишь написать код, обрабатывающий событие `KeyDown`, явно идентифицировать требуемую клавишу и при помощи оператора `Unload` или `End` закрыть форму. У процедуры обработки события `KeyDown` имеются два параметра. Первый из них возвращает код нажатой клавиши клавиатуры, а второй идентифицирует нажатую клавишу-модификатор. Для кода клавиши <Esc> в VBA имеется специальная константа `vbKeyEscape`. Следующий код (листинг 7.4) из модуля формы закрывает окно при нажатии клавиши <Esc>.

Листинг 7.4. Закрытие формы при нажатии клавиши <Esc>

```
Private Sub UserForm_KeyDown(ByVal KeyCode As MSForms.ReturnInteger, _
                             ByVal Shift As Integer)
   If KeyCode = vbKeyEscape Then
      Unload Me
   End If
End Sub
```

Подтверждение на закрытие окна

В проектах часто возникает ситуация, когда перед закрытием окна требуется получить подтверждение со стороны пользователя. Подобного эффекта можно избежать, воспользовавшись процедурой обработки события `QueryClose`,

которое генерируется непосредственно перед закрытием окна. У этой процедуры имеются два параметра. Если первый из них принимает значение −1, то закрытия не происходит, если же 0, то окно закрывается. Второй параметр идентифицирует причину, вызвавшую закрытие окна. Например, в следующем коде (листинг 7.5) при закрытии пользовательского окна генерируется диалоговое окно с двумя кнопками **Yes** и **No**, требующее подтверждение со стороны пользователя. Если он нажмет кнопку **Yes**, то окно закрывается, а если **No** — то не закрывается.

Листинг 7.5. Подтверждение на закрытие окна

```
Private Sub UserForm_QueryClose(Cancel As Integer, CloseMode As Integer)
   Select Case MsgBox("Закрыть окно?", vbYesNo + vbQuestion)
      Case vbYes
         Cancel = 0
      Case vbNo
         Cancel = -1
   End Select
End Sub
```

Задание местоположения формы

Первоначальное местоположение формы устанавливается свойством `StartUpPosition`. Допустимые его значения перечислены в табл. 7.7.

***Таблица 7.7.** Значения свойства `StartUpPosition`*

Значение	Описание
0	Координата левого верхнего угла формы устанавливается при помощи свойств `Top` и `Left`
1	По центру окна
2	Центрируется по окну экрана
3	В верхнем левом углу экрана

Например, следующий код (листинг 7.6) отображает форму так, чтобы ее левый верхний угол располагался в точке (100, 100).

Листинг 7.6. Задание местоположения формы

```
Private Sub UserForm_Initialize()
    Me.StartUpPosition = 0
```

```
    Me.Top = 100
    Me.Left = 100
End Sub
```

Модальная форма

Мономодальное окно — это такое окно, не закрыв которое нельзя отобразить на экране другое окно. По умолчанию пользовательское окно в VBA является мономодальным. Задать тип окна (мономодальный или полимодальный) можно при помощи необязательного параметра *style* метода Show.

```
Show style
```

Здесь параметр *style* имеет следующие допустимые значения: vbModal или 1 — мономодальная форма, vbModeless или 0 — полимодальная форма.

Например, при отображении формы на рабочем листе следующей инструкцией окно отображается в полимодальном режиме, и поэтому при открытом окне пользователь имеет доступ к ячейкам рабочего листа.

```
UserForm1.Show vbModeless
```

При запуске же окна следующей инструкцией оно имеет мономодальный тип, и поэтому ячейки рабочего листа недостижимы для пользователя до тех пор, пока он не закроет окно.

```
UserForm1.Show vbModal
```

Использование нескольких форм

В проекте может быть использовано несколько форм. При замене одной формы другой надо учитывать, в каком режиме (мономодальном или полимодальном) она открыта. Например, создайте две формы UserForm1 и UserForm2. На рабочем листе создайте кнопку. Установите значение ее свойства Name равным cmdForm1. При нажатии этой кнопки на экране будет отображаться окно первой формы.

Следующие два листинга (7.7 и 7.8) демонстрируют, как должны различаться коды для вызова второй формы при щелчке на первой, в зависимости от того, какой тип окна задан: мономодальный или полимодальный.

В мономодальном режиме, прежде чем отображать вторую форму, в коде приходится закрыть первую. При этом на экране всегда отображается только одна форма — либо первая, либо вторая.

Листинг 7.7, *а*. Мономодальный режим. Код из модуля рабочего листа

```
Private Sub cmdForm1_Click()
   UserForm1.Show vbModal
End Sub
```

Листинг 7.7, *б*. Мономодальный режим. Код из модуля `UserForm1`

```
Private Sub UserForm_Click()
   Unload UserForm1
   UserForm2.Show
End Sub
```

В полимодальном режиме нет нужды закрывать первую форму, и после щелчка на первой форме на экране отображаются обе формы. Для того чтобы обе формы были видны одновременно, вторую форму выводят с небольшим сдвигом относительно первой.

Листинг 7.8, *а*. Полимодальный режим. Код из модуля рабочего листа

```
Private Sub cmdForm1_Click()
   UserForm1.Show vbModeless
End Sub
```

Листинг 7.8, *б*. Полимодальный режим. Код из модуля `UserForm1`

```
Private Sub UserForm_Click()
   UserForm2.StartUpPosition = 0
   UserForm2.Top = UserForm1.Top + 20
   UserForm2.Left = UserForm1.Left + 20
   UserForm2.Show
End Sub
```

"Пасхальное яйцо"

"Пасхальное яйцо" (easter egg) — это скрытое диалоговое окно в приложении, в котором содержится информация о его создателе. Включение "пасхального яйца" в приложение является наиболее простым способом подтверждения авторства на программный продукт. В приводимом ниже примере (листинг 7.9) "пасхальное яйцо" отображается на экране только в случае, если пользователь щелкнет правой кнопкой мыши в области формы, лежащей в ее правом нижнем углу и занимающей одну девятую часть пользовательской области. Естественно, что до "пасхального яйца" сможет докопаться только тот, кто создавал приложение (рис. 7.5).

Для идентификации точки щелчка воспользуемся процедурой обработки события `MouseDown` (нажатие кнопки мыши) формы, которая имеет следующий синтаксис:

```
Private Sub object_MouseDown(ByVal Button As Long, ByVal Shift As Long,
↳ByVal X As Long, ByVal Y As Long)
```

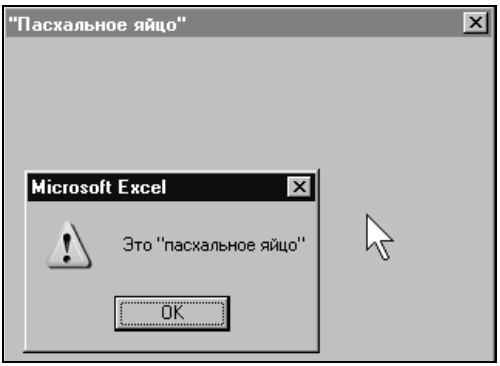


Рис. 7.5. "Пасхальное яйцо"

- *Button* — идентифицирует нажатую кнопку мыши. Допустимыми значениями могут быть следующие константы XlMouseButton: xlNoButton, xlPrimaryButton, xlSecondaryButton и xlMiddleButton;
- *Shift* — определяет нажатую клавишу-модификатор (<Shift>, <Ctrl> и <Alt>). Возвращает 0, если ни одна клавиша-модификатор не была нажата; возвращает 1, если была нажата клавиша <Shift>; возвращает 2, если была нажата клавиша <Ctrl>; возвращает 4, если была нажата клавиша <Alt>. Если была нажата комбинация клавиш, то возвращается сумма чисел-идентификаторов этих клавиш;
- *X*, *Y* — координаты точки, в которой была нажата клавиша мыши.

Листинг 7.9. "Пасхальное яйцо"

```
Private Sub UserForm_MouseDown(ByVal Button As Integer, _
      ByVal Shift As Integer, ByVal X As Single, ByVal Y As Single)
  Dim IsRight As Boolean
  IsRight = (Button = xlSecondaryButton) _
            And (X > 2 * Me.InsideWidth / 3) _
            And (Y > 2 * Me.InsideHeight / 3)
  If IsRight Then
     MsgBox "Это ""пасхальное яйцо""", vbExclamation
  End If
End Sub
```

Элементы управления

В VBA имеется обширный набор встроенных элементов управления, которые можно использовать как на рабочем листе, так и в форме. Применяя их, нетрудно создать любой пользовательский интерфейс, который будет

удовлетворять всем требованиям, предъявляемым к интерфейсу в среде Windows. Элементы управления создаются при помощи панели элементов **Toolbox** (рис. 7.6), которая отображается на экране либо выбором команды **View | Toolbox**, либо нажатием кнопки панели инструментов **Standard**. На панели **Toolbox** представлены кнопки, при помощи которых конструируются элементы управления. Все кнопки панели инструментов, за исключением кнопки **Выбор объекта**, служат для создания элементов управления. Щелкнув на кнопке **Выбор объекта** можно выбрать уже созданный в форме элемент управления для последующего его редактирования (изменения размеров или перемещения).

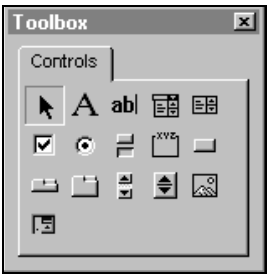


Рис. 7.6. Панель элементов **Toolbox**

Примечание

Панель инструментов **Элементы управления**, которая отображается MS Excel для создания элементов управления на рабочем листе, имеет меньший набор объектов, чем панель элементов из редактора Visual Basic. В ней нет рамок, набора вкладок и страниц.

Создание элементов управления на рабочем листе или форме, как правило, происходит на начальном этапе конструирования приложения. Иногда используется программное их создание в процессе работы приложения. Но этот подход применяется реже. В табл. 7.8 приведен список основных элементов управления и соответствующих кнопок панели элементов.

Таблица 7.8. *Элементы управления из панели элементов*

Элемент управления	Имя	Кнопка, его создающая
Поле	`TextBox`	
Надпись	`Label`	
Кнопка	`CommandButton`	
Список	`ListBox`	

Таблица 7.8 *(окончание)*

Элемент управления	Имя	Кнопка, его создающая
Поле со списком	`ComboBox`	
Полоса прокрутки	`ScrollBar`	
Счетчик	`SpinButton`	
Переключатель	`OptionButton`	
Флажок	`CheckBox`	
Выключатель	`ToggleButton`	
Рамка	`Frame`	
Рисунок	`Image`	
Набор страниц	`MultiPage`	
Набор вкладок	`TabStrip`	

Размещение элемента управления на форме

Для размещения элемента управления на форме нажмите соответствующую кнопку панели инструментов **Toolbox** и с помощью мыши перетащите рамку элемента управления в нужное место. После этого элемент управления можно перемещать, изменять его размеры, копировать в буфер обмена, вставлять из буфера обмена и удалять из формы.

Надпись (*Label*)

Элемент управления **Надпись** (Label) используется для отображения надписей, например заголовков элементов управления, не имеющих свойства `Caption`. Пользователь не может изменять текст, отображаемый в надписи во время выполнения программы. Основным свойством надписи является свойство `Caption`, устанавливающее отображаемый в ней текст.

Следующий пример демонстрирует различные типы надписей: простую, с рисунком, с рамкой и переносом слов (рис. 7.7). Для реализации этого проекта создайте форму, в которой расположите три надписи. Кроме того, необходим файл с рисунком. В данном случае это C:\flags\Flgrus.ico с изо-

бражением российского флага. В модуле формы наберите следующий код (листинг 7.10), который на этапе инициализации формы и задаст ее параметры. Рисунок в надпись загружается свойством `Picture`. Свойство `PicturePosition` определяет взаимное расположение рисунка и текста. Свойство `BorderStyle` устанавливает, отображается надпись с границей или без нее. Свойство же `WordWrap` задает автоматический перенос слов, если они не умещаются в одну строку.

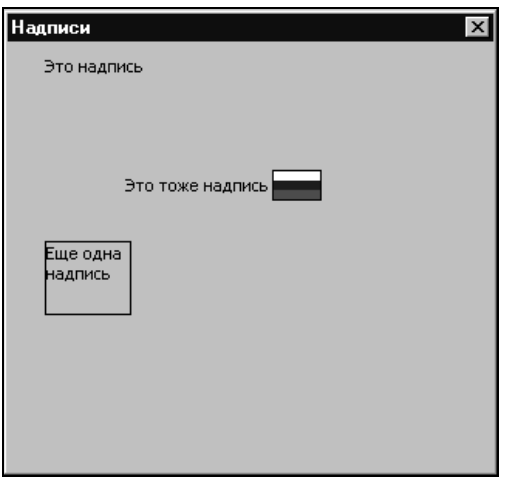


Рис. 7.7. Примеры надписей

Листинг 7.10. Примеры надписей

```
Private Sub UserForm_Initialize()
   Me.Caption = "Надписи"
   Label1.Caption = "Это надпись"
   Label2.Caption = "Это тоже надпись"
   Label2.Picture = LoadPicture("c:\flags\Flgrus.ico")
   Label2.PicturePosition = fmPicturePositionRightCenter
   Label3.BorderStyle = fmBorderStyleSingle
   Label3.WordWrap = True
   Label3.Caption = "Еще одна надпись"
End Sub
```

Поле (*TextBox*)

Элемент управления **Поле** или **Поле ввода** (TextBox) в основном применяется для ввода пользователем текста, который в последующем используется в программе, или для вывода в него результатов расчетов в программе.

Текст, введенный в поле, в коде может быть преобразован либо в числа, либо в формулы. Основным событием, связанным с полем ввода, является событие `Change`. В табл. 7.9 перечислены основные свойства поля ввода.

Таблица 7.9. *Основные свойства поля ввода*

Свойство	Описание
`Text`	Возвращает текст, содержащийся в поле
`Multiline`	Параметр, принимающий логические значения, который устанавливает многострочный режим ввода текста в поле
`ScrollBars`	Устанавливает режим отображения в поле полос прокрутки. Допустимые значения: `fmScrollBarsNone` (не выводить полос прокрутки), `fmScrollBarsHorizontal` (выводить горизонтальную полосу прокрутки), `fmScrollBarsVertical` (выводить вертикальную полосу прокрутки), `fmScrollBarsBoth` (выводить горизонтальную и вертикальную полосы прокрутки)
`SelLenght`, `SelStart`, `SelText`	Эти свойства характеризуют выделенный в поле фрагмент текста (длина, начало и сам фрагмент текста)
`MaxLength`	Устанавливает максимально допустимое количество вводимых в поле символов. Если это свойство равно 0, то нет ограничений на вводимое количество символов
`PasswordChar`	Устанавливает символ, отображаемый при вводе пароля. Если это свойство определено, то вместо вводимых символов в поле будет отображаться установленный символ

Сложение двух чисел

В качестве примера работы с полями ввода создадим демонстрационный проект, в котором по введенным в поля двум числам находится их сумма, а результат отображается в третьем поле. Итак, создайте форму, на которой расположите три надписи, три поля ввода и две кнопки (рис. 7.8), и, используя окно **Properties**, установите им значения свойств, как показано в табл. 7.10.

Таблица 7.10. *Значения свойств, установленные в окне* ***Properties***

Объект	Свойство	Значение
Форма	`Caption`	`c=a+b`
Надпись	`Caption`	`a`
Поле ввода	`Name`	`txtA`

Таблица 7.10 *(окончание)*

Объект	Свойство	Значение
Надпись	Caption	b
Поле ввода	Name	txtB
Надпись	Caption	c
Поле ввода	Name	txtC
Кнопка	Name	cmdOK
	Caption	OK
Кнопка	Name	cmdCancel
	Caption	Cancel

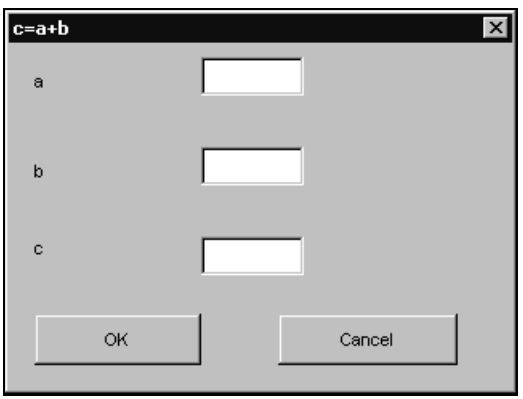


Рис. 7.8. Сложение двух чисел

В модуле формы наберите следующий код (листинг 7.11). Он обеспечит нахождение суммы двух чисел, введенных в поля `a` и `b`, и вывод результата в поле `c` при нажатии кнопки **OK**. Нажатие же кнопки **Cancel** приведет к закрытию окна.

Листинг 7.11. Сложение двух чисел

```
Private Sub cmdOK_Click()
   Dim a As Double, b As Double, c As Double
   a = txtA.Text
   b = txtB.Text
   c = a + b
```

```
   txtC.Text = c
End Sub

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

Кнопка с "горячей" клавишей

Свойство `Accelerator` элемента управления назначает буквенную или цифровую клавишу, при нажатии на которую одновременно с клавишей <Alt> выполняется процедура обработки события — щелчок на элементе управления. При этом буква или цифра на этой клавише должна входить в строку, задающую значение свойства `Caption` элемента управления, и эта буква или цифра будет отображаться подчеркнутой. Например, если в код из предыдущего *разд. "Сложение двух чисел"* добавить следующую процедуру инициализации формы (листинг 7.12), то кнопки **OK** и **Cancel** превратятся в **OK** и **Cancel**, а нажатие комбинаций клавиш <Alt>+<O> и <Alt>+<C> будет равносильно нажатию этих кнопок.

Листинг 7.12. Кнопка с "горячей" клавишей

```
Private Sub UserForm_Initialize()
   cmdOK.Accelerator = "O"
   cmdCancel.Accelerator = "C"
End Sub
```

Клавиши <Enter> и <Esc>

Свойство `Default` при значении, равном `True`, назначает кнопку, для которой генерируется событие `Click`, если пользователь нажмет клавишу <Enter>. Свойство же `Cancel` при значении, равном `True`, назначает кнопку, для которой генерируется событие `Click`, если пользователь нажмет клавишу <Esc>. Например в следующем коде (листинг 7.13) для проекта из *разд. "Сложение двух чисел"* клавишам <Enter> и <Esc> назначаются функции кнопок **OK** и **Cancel**, т. е. при нажатии клавиши <Enter> будет найдена искомая сумма, а клавиши <Esc> — закрыто окно.

Листинг 7.13. Кнопка с "горячей" клавишей

```
Private Sub UserForm_Initialize()
   cmdOK.Default = True
   cmdCancel.Cancel = True
End Sub
```

Суммирование с блокировкой результата для пользователя

Свойство `Enabled` при значении равным `False` полностью блокирует для пользователя элемент управления, так что элемент управления даже не может получать фокус. Свойство `Lock` при значении равным `True`, запрещает пользователю производить любые изменения в поле ввода, но, тем не менее, поле ввода может получать фокус, так что пользователь теперь имеет возможность копировать содержание поля ввода в буфер обмена. Следующий код (листинг 7.14) полностью блокирует для пользователя поле **с**, в которое выводится найденная сумма в проекте из *разд. "Сложение двух чисел"*.

Листинг 7.14. Суммирование с блокировкой результата для пользователя

```
Private Sub UserForm_Initialize()
   txtC.Enabled = False
End Sub
```

Как сделать, чтобы по нажатии на кнопке она не получала фокус

При нажатии на кнопке она получает фокус, и поэтому для повторного ввода значения в поле ввода оно должно сначала этот фокус получить. Можно ли сделать так, чтобы при нажатии кнопки она не получала фокус, т. е. фокус оставался бы на том же элементе, который имел его до нажатия. Ответ утвердительный. Свойство `TakeFocusOnClick` при значении, равном `False`, блокирует передачу фокуса. Например, для проекта из *разд. "Сложение двух чисел"* такую блокировку можно сделать следующей процедурой (листинг 7.15).

Листинг 7.15. Кнопка не получает фокус даже при ее нажатии

```
Private Sub UserForm_Initialize()
   cmdOK.TakeFocusOnClick = False
End Sub
```

Перемещение фокуса между полями при нажатии клавиши <Enter>

Для перемещения фокуса между полями по нажатии клавиши <Enter> у полей ввода надо обработать событие `KeyDown` для того, чтобы идентифицировать нажатую клавишу, и если таковой является клавиша <Enter>, то мето-

дом SetFocus следует установить фокус на требуемом поле ввода. В следующем примере (листинг 7.16) для проекта из *разд. "Сложение двух чисел"* при нажатии клавиши <Enter>, когда фокус имеет поле a, фокус перемещается на поле b. При нажатии же клавиши <Enter>, когда фокус имеет поле b, фокус перемещается на поле a, и, кроме того, выполняется процедура обработки события Click кнопки **OK**, т. е. находится искомая сумма. Поле c заблокировано для пользователя, но оно может получать фокус.

Листинг 7.16. Перемещение фокуса между полями при нажатии клавиши <Enter>

```
Private Sub txtA_KeyDown(ByVal KeyCode As MSForms.ReturnInteger, _
                         ByVal Shift As Integer)
   If KeyCode = vbKeyReturn Then
       txtB.SetFocus
   End If
End Sub

Private Sub txtB_KeyDown(ByVal KeyCode As MSForms.ReturnInteger, _
                         ByVal Shift As Integer)
   If KeyCode = vbKeyReturn Then
      cmdOK_Click
      txtA.SetFocus
   End If
End Sub

Private Sub UserForm_Initialize()
   txtC.Locked = True
End Sub
```

Всплывающая подсказка

Элементам управления можно устанавливать всплывающую подсказку при помощи свойства ControlTipText, которая появляется в виде окна с пояснительным текстом, отображаемым рядом с элементом управления, на который указывает указатель мыши. В следующем примере (листинг 7.17) для проекта из *разд. "Сложение двух чисел"* трем полям ввода и двум кнопкам задаются всплывающие подсказки (рис. 7.9).

Листинг 7.17. Всплывающие подсказки

```
Private Sub SetToolTips()
   txtA.ControlTipText = "Первое слагаемое"
   txtB.ControlTipText = "Второе слагаемое"
```

```
    txtC.ControlTipText = "Сумма двух чисел"
    cmdOK.ControlTipText = "Нахождение суммы"
    cmdCancel.ControlTipText = "Закрытие окна"
End Sub
```

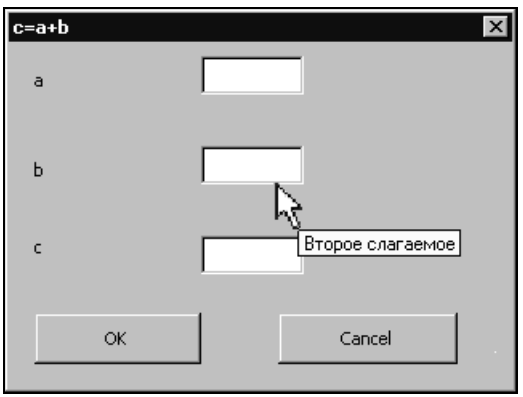


Рис. 7.9. Всплывающая подсказка

Поле вода пароля

Элемент управления **Textbox** позволяет создавать поле ввода пароля (password). Символы, которые отображаются в нем как набираемые, называются *эхо-символами* и устанавливаются свойством `PasswordChar`.

Рассмотрим пример простого приложения ввода пароля. Имеется поле ввода пароля и кнопка. Кнопка при инициализации окна заблокирована. Если пользователь введет в поле ввода правильный пароль (в данном случае `Winnie` в честь Винни-Пуха), то кнопка разблокируется, если же неправильный, то она повторно блокируется (рис. 7.10).

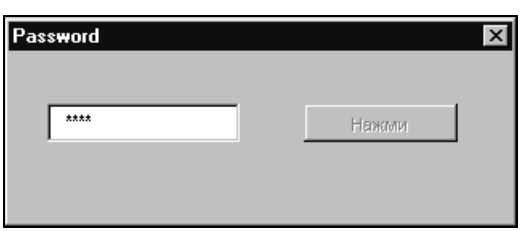


Рис. 7.10. Окно **Password**

Итак, создайте форму, на которой расположите поле ввода и кнопку, и, используя окно **Properties**, установите им значения свойств, как показано в табл. 7.11.

Таблица 7.11. *Значения свойств, установленные в окне* ***Properties***

Объект	Свойство	Значение
Форма	Caption	Password
Поле ввода	Name	txtPass
Кнопка	Name	cmdMsg
	Caption	Нажми

В модуле формы наберите следующий код (листинг 7.18). Для упрощения ввода пароля в нем не учитывается различие между регистрами букв. Это обеспечивается переводом всех введенных букв в нижний регистр функцией LCase.

Листинг 7.18. Ввод пароля

```
Private pass As String

Private Sub UserForm_Initialize()
   cmdMsg.Enabled = False
   txtPass.PasswordChar = "*"
   pass = "Winnie"
End Sub

Private Sub txtPass_KeyDown(ByVal KeyCode As MSForms.ReturnInteger, _
                            ByVal Shift As Integer)
   If KeyCode = vbKeyReturn Then
       If LCase(txtPass.Text) = LCase(pass) Then
           cmdMsg.Enabled = True
       Else
          cmdMsg.Enabled = False
       End If
   End If
End Sub

Private Sub cmdMsg_Click()
   MsgBox "Пароль введен правильно"
End Sub
```

Многострочное поле ввода

Поле ввода можно использовать как многострочное. Для этого надо установить значение свойства MultiLine равным True. В качестве примера сконструируем простой проект, который производит пересчет рублей в доллары.

В этом проекте имеются два поля ввода: простое и многострочное. В простое поле вводится исходная сумма, которая по нажатии клавиши <Enter> переводится в доллары, а результат отображается в многострочном поле (рис. 7.11).

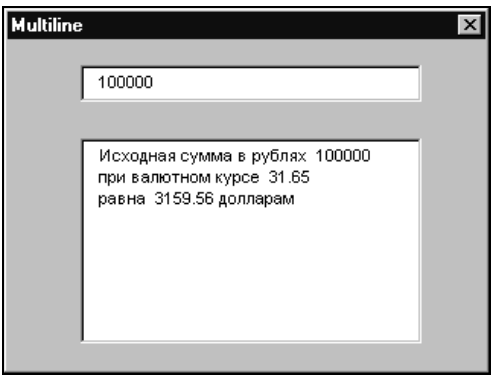


Рис. 7.11. Многострочное поле ввода

Итак, создайте форму, на которой расположите два поля ввода и кнопку, и, используя окно **Properties**, установите им значения свойств, как показано в табл. 7.12.

Таблица 7.12. *Значения свойств, установленные в окне* ***Properties***

Объект	Свойство	Значение
Поле ввода	Name	txtMoney
Поле ввода	Name	txtResult

В модуле формы наберите следующий код (листинг 7.19). С тем, чтобы пользователь не мог подкорректировать результаты расчетов, многострочное поле для него в проекте блокируется на этапе инициализации формы.

Листинг 7.19. Многострочное поле ввода

```
Private rate As Double

Private Sub UserForm_Initialize()
   rate = 31.65
   txtResult.MultiLine = True
   txtResult.Locked = True
End Sub
```

```
Private Sub txtMoney_KeyDown(ByVal KeyCode As MSForms.ReturnInteger, _
                             ByVal Shift As Integer)
   If KeyCode = vbKeyReturn Then
      Dim r As Double
      Dim d As Double
      r = txtMoney.Text
      d = FormatNumber(r / rate, 2)
      txtResult.Text = "Исходная сумма в рублях  " & r & vbCr & _
                       "при валютном курсе  " & rate & vbCr & _
                       "равна  " & d & " долларам"
   End If
End Sub
```

Обмен значениями между формами

В проекте может присутствовать не одна, а несколько форм. Как они могут обмениваться значениями? Ответ — посредством открытых переменных, причем они должны быть объявлены в стандартном модуле. Как это делается, продемонстрируем на следующем примере (листинг 7.20). Создайте две формы. В первой из них разместите два поля ввода и кнопку. Во второй — поле ввода. При нажатии на кнопку первой формы считываются значения из ее полей, затем они складываются, первая форма закрывается, вторая отображается, а найденная сумма выводится в ее поле ввода.

Листинг 7.20, *а*. Модуль первой формы

```
Private Sub CommandButton1_Click()
   res = CDbl(TextBox1.Text) + CDbl(TextBox2.Text)
   Unload Me
   UserForm2.Show
End Sub
```

Листинг 7.20, *б*. Модуль второй формы

```
Private Sub UserForm_Initialize()
   TextBox1.Text = res
End Sub
```

Листинг 7.20, *в*. Стандартный модуль

```
Public res As Double
```

Флажок (*CheckBox*) и Выключатель (*ToggleButton*)

Флажок и выключатель предоставляет пользователю возможность выбора. Основным свойством этих элементов управления является свойство `Value`, возвращающее их состояние. Эти элементы управления обычно имеют два состояния: установлен (значение свойства `Value` равно `True`) и сброшен (значение свойства `Value` равно `False`), но могут настраиваться на выбор из трех альтернатив при помощи свойства `TripleState`.

Управление видимостью элементов управления

Свойство `Visible` контролирует видимость элемента управления. Если его значение равно `True`, то он видим, а если `False`, то невидим. Следующий пример демонстрирует, как с помощью флажка можно контролировать видимость элемента управления (в данном случае поля ввода). Итак, создайте форму и в модуле кода наберите приводимый ниже код (листинг 7.21). Если флажок **Показать** установлен, то поле ввода отображается, а если снят, то оно скрыто (рис. 7.12).

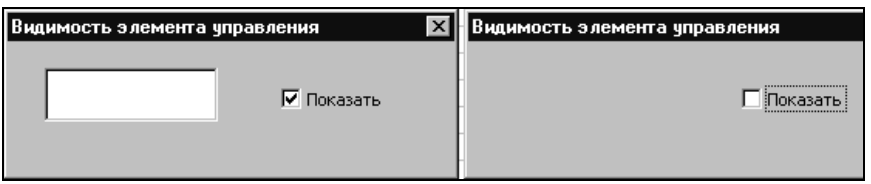


Рис. 7.12. Управление видимостью элементов управления

Листинг 7.21. Управление видимостью элементов управления

```
Private Sub UserForm_Initialize()
   CheckBox1.Caption = "Показать"
   TextBox1.Visible = True
   CheckBox1.Value = True
End Sub

Private Sub CheckBox1_Change()
   If CheckBox1.Value Then
      TextBox1.Visible = True
   Else
      TextBox1.Visible = False
  End If
End Sub
```

Управление достижимостью для пользователя элементов управления

Свойство `Enabled` управляет достижимостью для пользователя элементов управления. Если значение свойства равно `True`, то элемент управления может получить фокус и быть достижимым для пользователя, а если `False`, то не может. Следующий пример демонстрирует, как с помощью флажка можно контролировать достижимость элемента управления (в данном случае, кнопки). Итак, создайте форму и в модуле кода наберите приводимый ниже код (листинг 7.22). Если флажок **Блокировка** установлен, то кнопка **Нажми** заблокирована, а если снят, то не заблокирована (рис. 7.13).

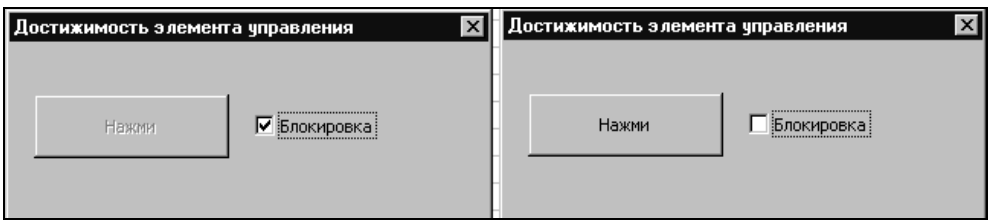


Рис. 7.13. Управление достижимостью для пользователя элементов управления

Листинг 7.22. Управление достижимостью для пользователя элементов управления

```
Private Sub UserForm_Initialize()
   CommandButton1.Caption = "Нажми"
   CheckBox1.Caption = "Блокировка"
   CheckBox1.Value = True
End Sub

Private Sub CheckBox1_Change()
   If CheckBox1.Value Then
      CommandButton1.Enabled = False
   Else
      CommandButton1.Enabled = True
   End If
End Sub

Private Sub CommandButton1_Click()
   MsgBox "Hello"
End Sub
```

Рамка (*Frame*)

Элемент управления **Рамка** (Frame) используется для визуальной группировки элементов управления. Основным свойством рамки является `Capture`, задающее надпись при рамке. Флажки и переключатели также в коде можно группировать при помощи свойства `GroupName`.

Переключатель

Элемент управления **Переключатель** (OptionButton) позволяет выбрать одну из нескольких взаимоисключающих альтернатив. Переключатели обычно отображаются группами по выбираемым альтернативам. Группировка производится при помощи элемента управления **Рамка** или свойства `GroupName` объекта `OptionButton`. Основными событиями переключателя являются события `Click` и `Change`, а основным свойством — свойство `Value`, возвращающее или устанавливающее его состояние. Если значение этого свойства равно `True`, то переключатель установлен, а если `False` — то сброшен.

Переключатель и выбор результирующей операции

В качестве примера использования переключателей слегка видоизменим проект из *разд. "Сложение двух чисел"*, а именно теперь будем находить не сумму, а результат по выбранной операции: сложения или вычитания. Исполняемая операция устанавливается за счет выбора соответствующего переключателя (рис. 7.14).

Итак, создайте форму, на которой расположите три надписи, три поля ввода и кнопку, а также рамку, в которой как в контейнере разместите два переключателя, и, используя окно **Properties**, установите им значения свойств, как показано в табл. 7.13.

Таблица 7.13. *Значения свойств, установленные в окне* ***Properties***

Объект	Свойство	Значение
Надпись	`Caption`	`a`
Поле ввода	`Name`	`txtA`
Надпись	`Caption`	`b`
Поле ввода	`Name`	`txtB`
Надпись	`Caption`	`c`
Поле ввода	`Name`	`txtC`
Кнопка	`Name`	`cmdOK`
	`Caption`	`OK`

Таблица 7.13 *(окончание)*

Объект	Свойство	Значение
Рамка	Caption	Операция
Переключатель	Name	optAdd
	Caption	Сложение
Переключатель	Name	optSub
	Caption	Сложение

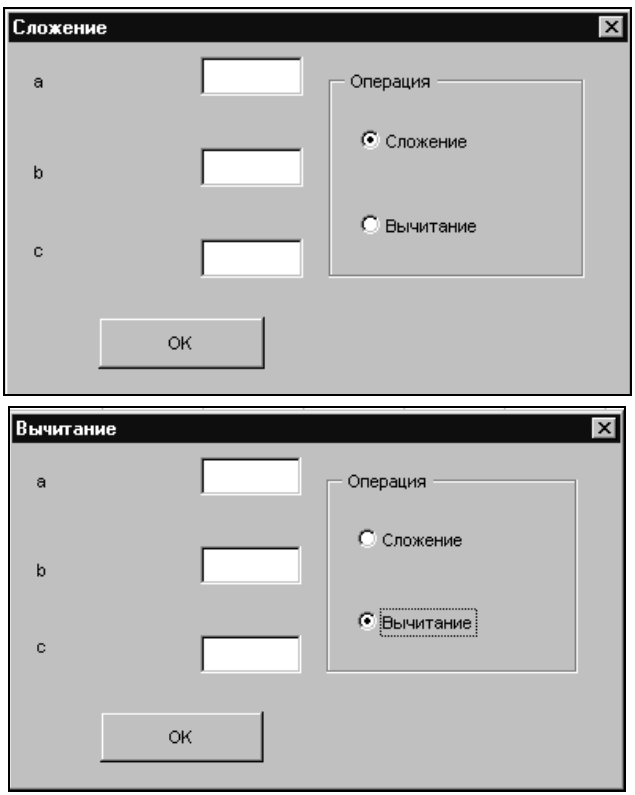


Рис. 7.14. Выбор результирующей операции

В модуле формы наберите следующий код (листинг 7.23). В процедуре обработки события `Click` кнопки происходит идентификация установленного переключателя и на основе этого производится расчет по соответствующей формуле. Процедуры обработки события `Click` переключателя выводят название выбранной операции в заголовок формы.

Листинг 7.23. Выбор результирующей операции

```
Private Sub UserForm_Initialize()
   optAdd.Value = True
End Sub

Private Sub cmdOK_Click()
   Dim a As Double, b As Double, c As Double
   a = txtA.Text
   b = txtB.Text
   If optAdd.Value Then
      c = a + b
   End If
   If optSub.Value Then
      c = a - b
   End If
   txtC.Text = c
End Sub

Private Sub optAdd_Click()
   Me.Caption = "Сложение"
End Sub

Private Sub optSub_Click()
   Me.Caption = "Вычитание"
End Sub
```

Полоса прокрутки (*ScrollBar*) и Счетчик (*SpinButton*)

Элемент управления **Полоса прокрутки** (ScrollBar) применяется для установки числового значения, причем этот элемент может устанавливать только целые неотрицательные значения. Основным событием элемента управления **Полоса прокрутки** является `Change`, а основными свойствами — свойства `Value`, `Min` и `Max`, устанавливающие соответственно текущее, минимальное и максимальное значения. Элемент управления **Счетчик** (SpinButton) по своим функциональным возможностям аналогичен полосе прокрутки, но у него нет ползунка.

Синхронизированная работа поля ввода и счетчика

Счетчик позволяет установить целое значение, которое можно затем вывести в поле ввода. Как сделать работу поля ввода и счетчика синхронизированными, чтобы текущее значение счетчика выводилось в поле, а число,

введенное в поле ввода, становилось значением счетчика (рис. 7.15). В этом случае, как у счетчика, так и у поля надо согласованно обработать события `Change`.

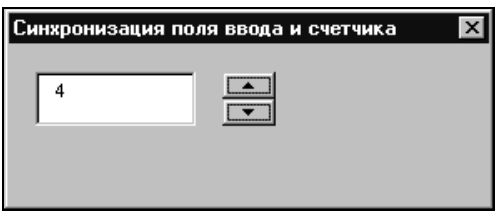


Рис. 7.15. Синхронизированная работа поля ввода и счетчика

Итак, создайте форму, в которой расположите поле и счетчик. В модуле формы наберите приводимый ниже код (листинг 7.24). При считывании данных из поля, прежде чем их присваивать значению `Value` счетчика, надо убедиться, являются ли эти данные числом, что можно сделать при помощи функции `IsNumeric`. Кроме того, необходима проверка, принадлежит ли число интервалу допустимых значений для счетчика, т. е. лежит ли оно в интервале, определяемом свойствами `Min` и `Max` счетчика. В данном примере значения этих свойств устанавливаются равными 1 и 5 при инициализации формы.

Листинг 7.24. Синхронизированная работа поля ввода и счетчика

```
Private Sub UserForm_Initialize()
   SpinButton1.Min = 1
   SpinButton1.Max = 5
End Sub

Private Sub SpinButton1_Change()
   TextBox1.Text = SpinButton1.Value
End Sub

Private Sub TextBox1_Change()
   Dim v As Variant
   v = TextBox1.Text
   If IsNumeric(v) Then
      If SpinButton1.Min <= v And v <= SpinButton1.Max Then
         SpinButton1.Value = v
      End If
   End If
End Sub
```

Усовершенствование элемента *Подбор параметра*

Элемент управления **Подбор параметра** позволяет на рабочем листе найти корень уравнения, но использование подбора параметра при решении уравнения требует выполнить относительно большой объем работы.

1. Представить уравнение в таком виде, чтобы неизвестное входило только в его левую часть, а правая часть была постоянной. Например, уравнение $x^2 = x + 1$ надо привести к виду $x^2 - x - 1 = 0$.
2. Выбрать команду **Сервис | Параметры** и на вкладке **Вычисления** появившегося диалогового окна **Параметры** в поле **Относительная погрешность** указать погрешность, с которой будет находиться корень уравнения. Например, 10^{-5}.
3. На рабочем листе отвести одну ячейку, например **B1**, под неизвестное. Ввести в эту ячейку начальное приближение к корню, например, 1.
4. Отвести на рабочем листе другую ячейку, например **B2**, под левую часть уравнения. Ввести в эту ячейку формулу левой части уравнения. В рассматриваемом случае формулу `=B1^2-B1-1`.
5. Выбрать команду **Сервис | Подбор параметра** и в появившемся диалоговом окне **Подбор параметра** в поле **Установить в ячейке** дать ссылку на ячейку, отведенную под левую часть уравнения. В нашем конкретном случае, **B2**. В поле **Значение** ввести число, стоящее в правой части уравнения. В данном случае надо ввести `0`. В поле **Изменяя значение ячейки** дать ссылку на ячейку, отведенную под неизвестное, в рассматриваемом примере **B1**. Нажать кнопку **OK**.
6. На экране отобразится диалоговое окно **Результат подбора параметра**, а в ячейке **B1** вместо начального приближения будет находиться найденное значение корня. В данном случае 1.618037.

Применение подбора параметра требует довольно большого объема работы. Давайте создадим приложение, облегчающую работу пользователя при решении уравнения.

Перейдем непосредственно к конструированию приложения. Создайте форму с четырьмя надписями, четырьмя полями ввода, счетчиком и кнопкой (рис. 7.16). Установите значения свойства `Name` элементов управления, как показано в табл. 7.14. В модуле формы наберите приводимый ниже код (листинг 7.25).

Таблица 7.14. *Значения свойства* `Name`*, установленные в окне* ***Properties***

Элемент управления	Значение свойства `Name`	Описание
Поле ввода	`txtBegin`	Пользователь вводит начальное приближение к корню

Таблица 7.14 *(окончание)*

Элемент управления	Значение свойства Name	Описание
Надпись	lblBegin	Пояснительная надпись для поля ввода txtBegin
Поле ввода	txtEquation	Пользователь вводит левую часть уравнения. Уравнение должно быть приведено к виду, когда его правая часть равна 0. Вводимая левая часть уравнения должна начинаться со знака равенства, записываться по правилам языка программирования и вместо неизвестного должен быть использован символ "x". Например, =x^2-x-1 В программе введенное выражение будет преобразовано в формулу рабочего листа
Надпись	lblEquation	Пояснительная надпись для поля ввода txtEquation
Поле ввода	txtRoot	Выводится найденное значение корня. Поле недостижимо для пользователя
Надпись	lblRoot	Пояснительная надпись для поля ввода txtRoot
Поле ввода	txtTol	Вводится относительная точность нахождения корня при помощи счетчика
Надпись	lblTol	Пояснительная надпись для поля ввода txtTol
Счетчик	spnTol	Задает относительную точность нахождения корня, которая выводится в поле txtTol
Кнопка	cmdOK	Считывает начальное приближение, левую часть уравнения и точность нахождения корня. Присваивает ячейке **B1** активного рабочего листа имя "x". Проверяет, преобразуется ли введенная левая часть уравнения в формулу рабочего листа. Если нет, то на экране отображается сообщение, и выполнение программы прерывается. Если данные введены корректно, то начальное приближение вводится в ячейку **B1** (ее имя теперь "x") рабочего листа. Формула левой части уравнения вводится в ячейку **B2**. При помощи метода **Подбор параметра** (метод GoalSeek диапазона) ищется корень. Если корень не найден, отображается сообщение. Если корень найден, то он предварительно форматируется с учетом введенной точности нахождения, а затем выводится в поле txtRoot

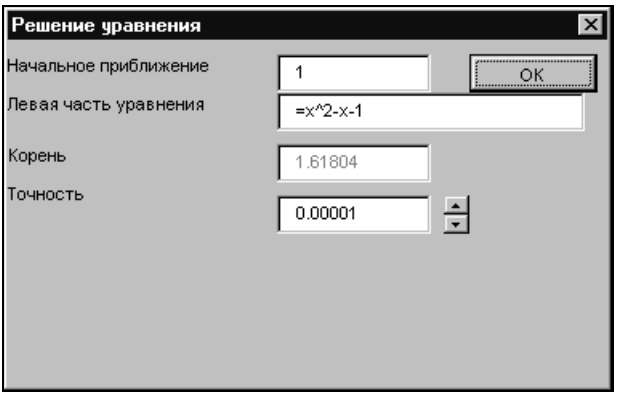


Рис. 7.16. Окно **Решение уравнения**

Листинг 7.25. Решение уравнения

```
Private Sub UserForm_Initialize()
   Me.Caption = "Решение уравнения"
   lblBegin.Caption = "Начальное приближение"
   lblEquation.Caption = "Левая часть уравнения"
   lblRoot.Caption = "Корень"
   lblTol.Caption = "Точность"
   cmdOK.Caption = "OK"
   txtRoot.Enabled = False
   txtTol.Locked = True
   With spnTol
      .Min = 1
      .Max = 7
      .Value = 3
   End With
   Range("B1").Name = "x"
End Sub

Private Sub spnTol_Change()
   With txtTol
      .Text = 0.1 ^ spnTol.Value
      Application.MaxChange = CDbl(.Text)
   End With
End Sub

Private Sub cmdOK_Click()
   Dim eq As String
   Dim x0 As Double
```

```
   Dim x As Double
   Dim check As Boolean
   x0 = CDbl(txtBegin.Text)
   eq = LCase(Trim(txtEquation.Text))
   Range("B1").Value = x0
   If IsError(Evaluate(eq)) Then
      MsgBox "Корректно введите функцию"
      txtEquation.SetFocus
      Exit Sub
   End If
   Range("B2").Value = eq
   If Not IsNumeric(Range("B2").Value) Then
      MsgBox "Корректно введите функцию"
      txtEquation.SetFocus
      Exit Sub
   End If
   check = Range("B2").GoalSeek(Goal:=0, ChangingCell:=Range("x"))
   If Not check Then
      MsgBox "Корень не найден"
      Exit Sub
   End If
   x = Range("x").Value
   txtRoot.Text = FormatNumber(x, spnTol.Value)
End Sub

Private Sub UserForm_Terminate()
   ActiveWorkbook.Names("x").Delete
   ActiveSheet.Range("B1:B2").Clear
End Sub
```

Список (*ListBox*)

Элемент управления **Список** (ListBox) применяется для хранения списка значений. В списке пользователь может выбрать одно или несколько значений, которые в последующем используются в тексте программы. Обратите внимание, что на этапе конструирования визуально список похож на поле ввода. Обычно выбор элемента из списка производится щелчком на элементе. Двойной же щелчок на элементе применяется для выполнения каких-то действий в программе, связанных с этим элементом.

Поэлементное заполнение списка

Заполнение списка поэлементно осуществляется методом `AddItem`. Для примера, создадим простой проект (листинг 7.26), в котором имеется форма со

списком. В список выведены названия городов. Выбор элемента из списка ни к чему не приводит, двойной же щелчок на элементе списка выводит его в очередную свободную ячейку первого столбца активного рабочего листа (рис. 7.17). Выбранное значение из списка возвращается свойством `Text`. Число же заполненных ячеек столбца возвращается свойством `CountA` объекта `WorksheetFunction`.

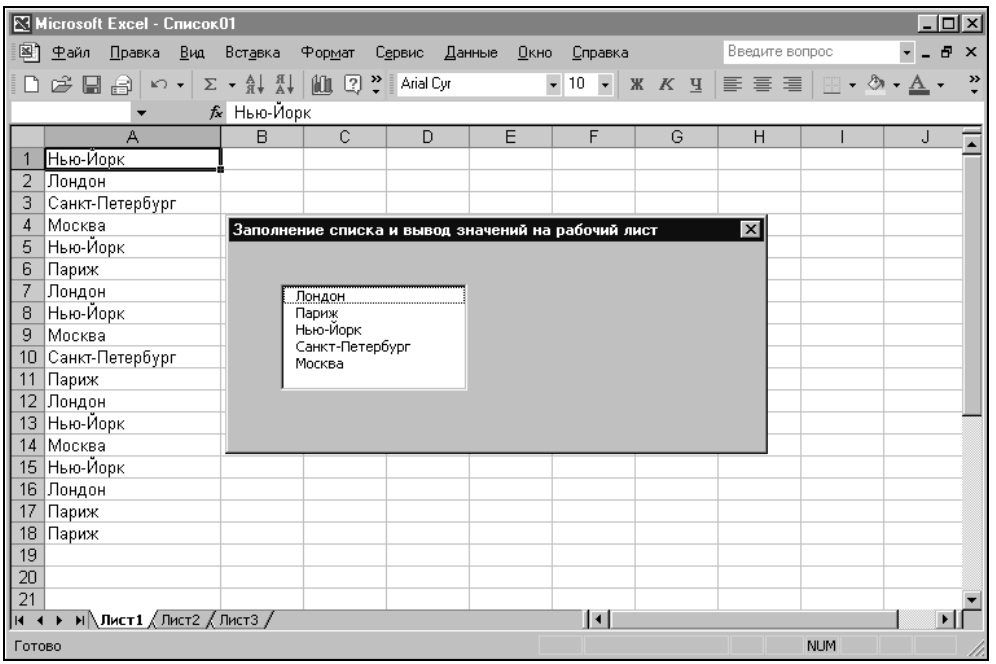


Рис. 7.17. Поэлементное заполнение списка и вывод значений на рабочий лист

Листинг 7.26. Заполнение списка и вывод значений на рабочий лист

```
Private Sub UserForm_Initialize()
   ListBox1.AddItem "Лондон"
   ListBox1.AddItem "Париж"
   ListBox1.AddItem "Нью-Йорк"
   ListBox1.AddItem "Санкт-Петербург"
   ListBox1.AddItem "Москва"
End Sub

Private Sub ListBox1_DblClick(ByVal Cancel As MSForms.ReturnBoolean)
   Dim n As Long
```

```
    n = Application.WorksheetFunction.CountA(Columns(1))
    Cells(n + 1, 1).Value = ListBox1.Text
End Sub
```

Заполнение списка из массива и выбор операции

Заполнять список можно поэлементно, а можно и за одну операцию. Для этого надо свойству `List` списка присвоить значение переменной, содержащей в себе массив значений. В качестве примера еще раз вернемся к проекту *"Сложение двух чисел"*. В этот раз будем находить не сумму двух чисел, а совершать над ними одну из четырех арифметических операций, перечисленных в списке (рис. 7.18).

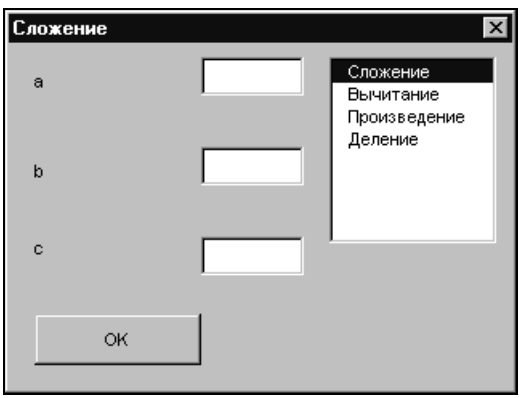


Рис. 7.18. Заполнение списка из массива и выбор операции

Итак, создайте форму, на которой расположите три надписи, три поля ввода, кнопку и список. У всех элементов управления, кроме списка, установите значения свойств при помощи окна **Properties**, как было описано в *разд. "Сложение двух чисел"*. У списка установите значения свойства `Name` равным `lstOp`. В модуле формы наберите приводимый ниже код (листинг 7.27). Элемент в списке идентифицируется по индексу, который возвращается свойством `ListIndex`. Индексация элементов начинается с 0. В данном коде по выбору элемента (другими словами, операции) из списка этот элемент выводится в заголовке формы. Это обеспечивается обработкой события `Click` списка.

Листинг 7.27. Заполнение списка из массива и выбор операции

```
Private Sub UserForm_Initialize()
    Dim v As Variant
    v = Array("Сложение", "Вычитание", "Произведение", "Деление")
```

```
   lstOp.List = v
   lstOp.ListIndex = 0
   txtC.Locked = True
End Sub

Private Sub cmdOK_Click()
   Dim a As Double, b As Double, c As Double
   a = txtA.Text
   b = txtB.Text
   Select Case lstOp.ListIndex
      Case 0
        c = a + b
      Case 1
        c = a – b
      Case 2
        c = a * b
      Case 3
         c = a / b
   End Select
   txtC.Text = c
End Sub

Private Sub lstOp_Click()
   Me.Caption = lstOp.Text
End Sub
```

Заполнение списка из диапазона

Свойство `RowSource` позволяет заполнять список из диапазона. Значением этого свойства является строка с именем диапазона. Продемонстрируем работу этого свойства на примере, причем рассмотрим как случай, когда диапазон, из которого берутся данные, известен заранее, так и ситуацию, когда известна единственная его ячейка.

Создадим простой проект по учету поездок сотрудников некоторой фирмы. Итак, имеется книга с тремя рабочими листами: **Города, Сотрудники** и **Поездки**.

- На листе **Города** в диапазоны **A1:A5** введен список городов, в которые намечены поездки сотрудников.
- На листе **Сотрудники** в столбец **A**, начиная с ячейки **A1**, введены фамилии сотрудников. Размер этого списка не фиксирован и может изменяться по усмотрению пользователя.
- На листе **Поездки** в два столбца выводится список поездок сотрудников по городам. Заполнение списка на листе **Поездки** производится с по-

мощью окна **Заполнение списка из массива** (рис. 7.19). В этом окне имеются два списка (с именами сотрудников и названиями городов) и кнопка. Списки заполняются из диапазонов соответствующих рабочих листов. Нажатие кнопки **OK** вызывает ввод выбранных данных в очередную пустую строку списка, расположенного на листе **Поездки**.

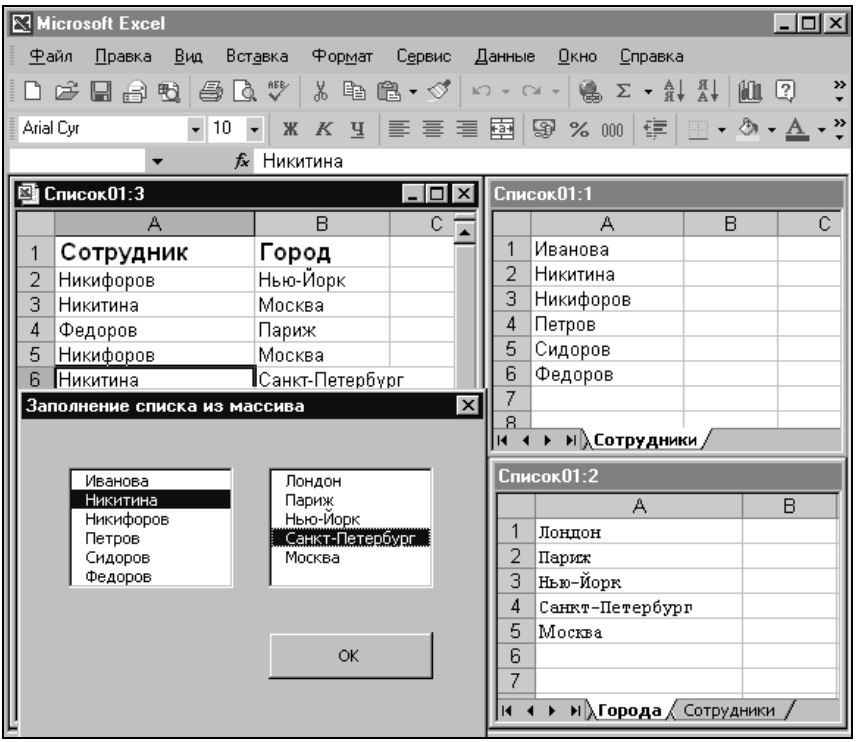


Рис. 7.19. Заполнение списка из диапазона

Создайте форму, на которой расположите два списка и кнопку, и, используя окно **Properties**, установите им значения свойств, как показано в табл. 7.15.

Таблица 7.15. *Значения свойств, установленные в окне* ***Properties***

Объект	Свойство	Значение
Список	Name	lstName
Список	Name	lstCity
Кнопка	Name	cmdOK
	Caption	OK

В модуле формы наберите приводимый ниже код (листинг 7.28). При инициализации формы производится заполнение списков. Список городов заполняется с явным указанием адреса диапазона, а прежде чем заполнить список сотрудников, т. к. по сценарию он может иметь переменный размер, происходит сначала идентификация диапазона, потом определение его адреса, а уже затем заполнение списка. При нажатии кнопки **OK** в первую очередь проверяется, были ли выбраны данные из списков, далее определяется номер первой свободной строки в списке из листа **Поездки**, и в нее выводятся выбранные данные из формы.

Листинг 7.28. Заполнение списка из диапазона

```
Private Sub UserForm_Initialize()
   lstCity.RowSource = "Города!A1:A5"
   Dim r As Range
   Set r = Worksheets("Сотрудники").Range("A1").CurrentRegion
   lstName.RowSource = r.Address(RowAbsolute:=False, _
                                 ColumnAbsolute:=False, _
                                 External:=True)
End Sub

Private Sub cmdOK_Click()
   If lstName.ListIndex < 0 Then
      MsgBox "lstName"
      Exit Sub
   End If
   If lstCity.ListIndex < 0 Then
      MsgBox "lstName"
      Exit Sub
   End If
   Dim r As Range
   Set r = Worksheets("Поездки").Range("A1").CurrentRegion
   Dim n As Long
   n = r.Rows.Count
   Worksheets("Поездки").Activate
   Cells(n + 1, 1).Value = lstName.Text
   Cells(n + 1, 2).Value = lstCity.Text
End Sub
```

Выбор нескольких элементов из списка

В списке можно выбирать как один, так и несколько элементов. Свойство `MultiSelect` устанавливает режим, при котором допустим такой выбор.

Допустимыми значениями свойства `MultiSelect` являются константы:

- `fmMultiSelectSingle` — разрешен выбор только одного элемента;
- `fmMultiSelectMulti` — щелчок на элементе или нажатие клавиши <Spacebar> выделяет элемент или снимает с него выделение;
- `fmMultiSelectExtended` — щелчок на элементе при нажатой клавише <Ctrl> выделяет элемент или снимает с него выделение. Щелчок на элементе при нажатой клавише <Shift> добавляет в выделение диапазон элементов от предыдущего выделенного до текущего.

Свойство `Selected` используется для идентификации выбранных элементов. Если его значение равно `True`, то элемент выбран, если равно `False`, то не выбран. Свойство `List` возвращает элемент с указанным индексом. При этом надо помнить, что индексация элементов производится с 0. Общее количество элементов списка возвращается свойством `ListCount`.

Продемонстрируем, как можно выбирать несколько элементов из списка на следующем проекте (рис. 7.20). Создайте форму, на которой расположите список и кнопку, и, используя окно **Properties**, установите им значения свойств, как показано в табл. 7.16.

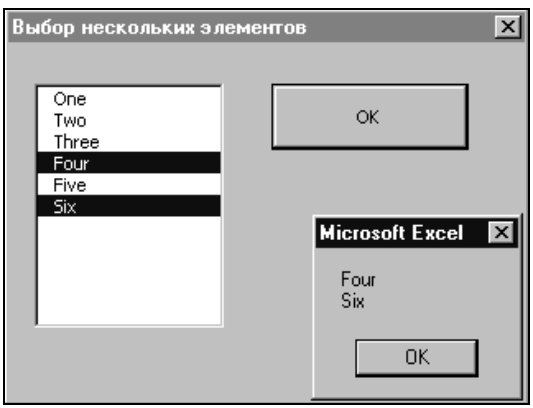


Рис. 7.20. Выбор нескольких элементов из списка

Таблица 7.16. *Значения свойств, установленные в окне* ***Properties***

Объект	Свойство	Значение
Список	`Name`	`lstNum`
Кнопка	`Name`	`cmdOK`
	`Caption`	`OK`

При инициализации формы производится ее заполнение и установка режима, позволяющего выбирать несколько элементов. Нажатие кнопки **OK** вызывает последовательное считывание выбранных элементов и составление на их основе информационной строки, которая отображается в диалоговом окне (листинг 7.29).

Листинг 7.29. Выбор нескольких элементов из списка

```
Private Sub UserForm_Initialize()
   lstNum.AddItem "One"
   lstNum.AddItem "Two"
   lstNum.AddItem "Three"
   lstNum.AddItem "Four"
   lstNum.AddItem "Five"
   lstNum.AddItem "Six"
   lstNum.MultiSelect = fmMultiSelectMulti
End Sub

Private Sub cmdOK_Click()
   Dim i As Integer
   Dim s As String
   For i = 0 To lstNum.ListCount - 1
      If lstNum.Selected(i) Then
         s = s + lstNum.List(i) & vbCr
      End If
   Next
   If s = Empty Then
      MsgBox "Nothing"
   Else
      MsgBox s
   End If
End Sub
```

Согласованная работа двух списков

Часто необходимо обеспечить согласованную работу двух списков. Например, некоторое издательство сотрудничает с определенными магазинами в разных городах (рис. 7.21). В один список выводится перечень городов. При выборе же элемента из этого списка во втором списке отображается перечень магазинов. Как этого добиться? Очень просто. Элементы массива могут иметь тип `Variant`, а переменной типа `Variant` может быть все, что угодно, в частности другой массив. Поэтому в VBA допустимо создание массива массивов, а дальше — элементарно, можно, например, поступить, как сделано в следующем коде (листинг 7.30).

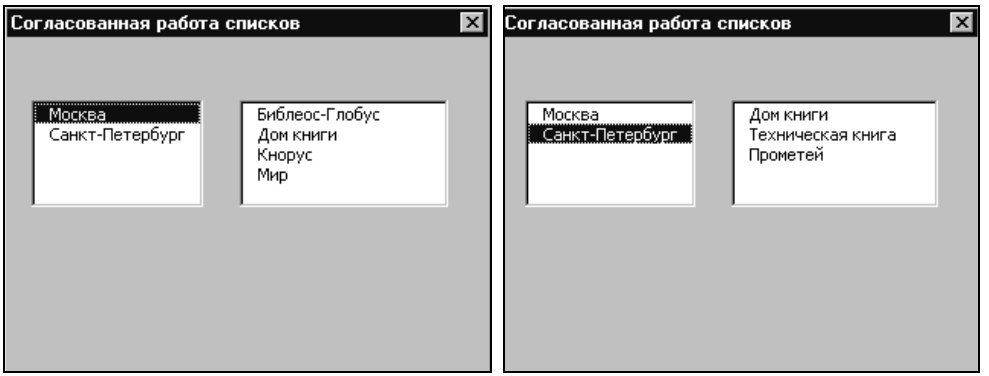


Рис. 7.21. Согласованная работа двух списков

Листинг 7.30. Согласованная работа двух списков

```
Dim city As Variant
Dim shop(2) As Variant

Private Sub UserForm_Initialize()
   city = Array("Москва", "Санкт-Петербург")
   shop(0) = Array("Библеос-Глобус", "Дом книги", "Кнорус", "Мир")
   shop(1) = Array("Дом книги", "Техническая книга", "Прометей")
   ListBox1.List = city
   ListBox1.ListIndex = 0
End Sub

Private Sub ListBox1_Click()
   ListBox2.Clear
   ListBox2.List = shop(ListBox1.ListIndex)
End Sub
```

Многостолбцовый список

Списки могут быть не только одностолбцовыми, но и многостолбцовыми. Для этого надо установить значение свойства `ColumnCount`, указывающее, сколько столбцов имеется в списке.

Кроме того, при помощи свойства `ColumnWidths` можно установить ширину каждого из столбцов списка.

```
ColumnWidths = String
```

Здесь параметр *`String`* представляет собой строку, образованную из чисел, которые равны ширине соответствующего столбца. Разделителем является

точка с запятой. Например, строка "`90;80`" говорит о том, что под первый столбец отводится 90 пунктов, а под второй — 80.

Доступ к элементам списка производится свойством `List`, первый параметр которого указывает его индекс, а второй — номер столбца. Нумерация как индексов, так и столбцов начинается с 0.

`List(`*`row`*`, `*`column`*`)`

В качестве примера сконструируем демонстрационный проект, в котором в двухстолбцовый список выводится заданное число случайных чисел (рис. 7.22). Создайте форму, на которой расположите список, поле ввода и кнопку, и, используя окно **Properties**, установите им значения свойств, как показано в табл. 7.17.

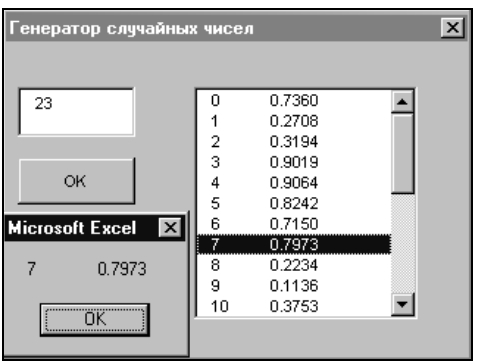


Рис. 7.22. Многостолбцовый список

Таблица 7.17. *Значения свойств, установленные в окне* ***Properties***

Объект	Свойство	Значение
Список	`Name`	`lstRnd`
Поле ввода	`Name`	`txtNum`
Кнопка	`Name`	`cmdOK`
	`Caption`	`OK`

При инициализации формы устанавливается двухстолбцовый режим списка, под первый столбец отводится 30 пунктов, а под второй — 50. При нажатии кнопки **OK** список сначала очищается методом `Clear`, а затем заполняется указанным числом случайных чисел с их номерами. При выборе элемента из списка отображается окно с сообщением о сделанном выборе (листинг 7.31).

Листинг 7.31. Многостолбцовый список

```
Private Sub UserForm_Initialize()
   lstRnd.ColumnCount = 2
   lstRnd.ColumnWidths = "30;50"
End Sub

Private Sub cmdOK_Click()
    lstRnd.Clear
    Dim i As Integer, n As Integer
    n = txtNum.Text
    If n <= 0 Then
        MsgBox "Число должно быть положительным"
        Exit Sub
    End If
    Randomize
    For i = 0 To n
       lstRnd.AddItem i
       lstRnd.List(i, 1) = FormatNumber(Rnd(), 4)
    Next
End Sub

Private Sub lstRnd_Change()
   MsgBox lstRnd.List(lstRnd.ListIndex, 0) & vbTab _
          & lstRnd.List(lstRnd.ListIndex, 1)
End Sub
```

Заполнение многостолбцового списка из диапазона и нахождение среднего значения выбранных чисел

Многостолбцовый список, так же как и одностолбцовый, можно заполнять из диапазона данных. Продемонстрируем это на примере приложения, вычисляющего средний балл у выбранной группы студентов из списка. Список заполняется данными из диапазона, верхним правым углом которого служит ячейка **A1**. Первый столбец содержит фамилии студентов, а второй — их оценки (рис. 7.23).

Перейдем к конструированию приложения. Создайте форму с надписью, полем ввода, списком и кнопкой. В модуле формы наберите приводимый ниже код (листинг 7.32). Заполнение списка и задание значения свойства `Caption` элементов управления и формы производится на этапе инициализации формы. При нажатии кнопки **OK** сначала проверяется, имеются ли выбранные элементы. Это делается потому, что средний балл равен сумме баллов, деленной на их количество. Если в списке не выбран ни один элемент, то данная формула приведет к генерации ошибки деления на ноль.

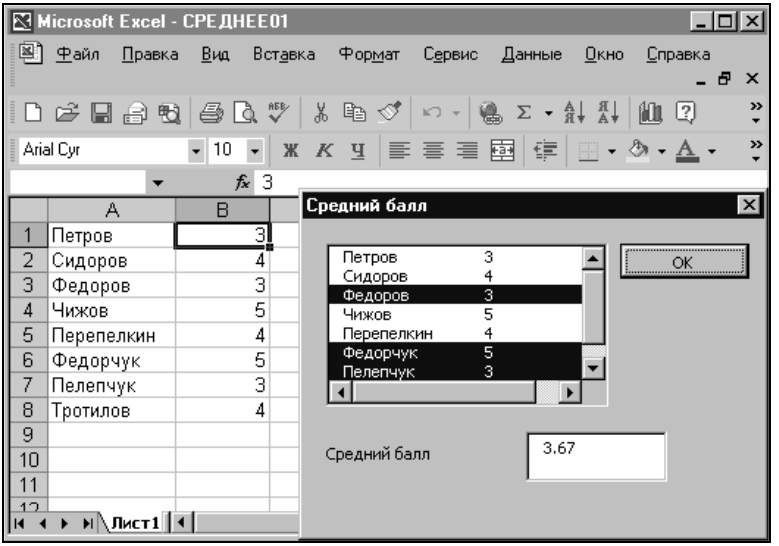


Рис. 7.23. Нахождение среднего балла

Листинг 7.32. Средний балл

```
Private Sub UserForm_Initialize()
   Dim r As Range
   Me.Caption = "Средний балл"
   CommandButton1.Caption = "OK"
   Label1.Caption = "Средний балл"
   Set r = Range("A1").CurrentRegion
   With ListBox1
      .ColumnCount = 2
      .RowSource = r.Address
      .MultiSelect = fmMultiSelectMulti
   End With
End Sub

Private Sub CommandButton1_Click()
   Dim s As Double
   Dim n As Integer, i As Integer
   n = 0
   For i = 0 To ListBox1.ListCount - 1
       If ListBox1.Selected(i) Then n = n + 1
   Next
   If n = 0 Then
      TextBox1.Text = Empty
```

```
      Exit Sub
   End If
   s = 0
   For i = 0 To ListBox1.ListCount - 1
       If ListBox1.Selected(i) Then s = s + ListBox1.List(i, 1)
   Next
   s = s / n
   TextBox1.Text = FormatNumber(s, 2)
End Sub
```

Скрытие данных в многостолбцовом списке

Если в значении свойства `ColumnWidths` какой-либо компонент указан нулевым, то соответствующий столбец не отображается. Это можно делать для скрытия некоей вспомогательной информации. Например, создадим следующий демонстрационный проект — отдел кадров. В нем в список на основе диапазона **A2:C9** рабочего листа **Сотрудники** вводится информация от сотрудниках фирмы: фамилия, должность и стаж работы (рис. 7.24). Хотя все эти данные загружаются в список, визуально в нем отображаются только фамилии. При щелчке на элементе списка отображается диалоговое окно с фамилией выбранного сотрудника. Установив требуемую комбинацию флажков **Должность** и **Стаж**, отображаемую о выбранном сотруднике информацию можно расширить. Для реализации этого проекта создайте книгу

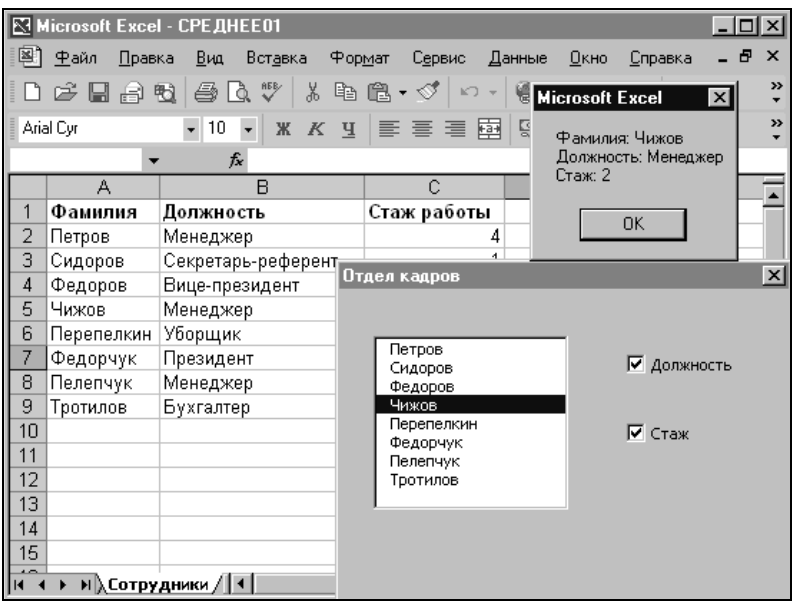


Рис. 7.24. Отдел кадров

с рабочим листом **Сотрудники**, в диапазон **A2:C9** которого введены данные о сотрудниках. Сконструируйте форму, на которой расположите список и два флажка. В модуле формы наберите приводимый ниже код (листинг 7.33). Заполнение списка и задание значения свойства `Caption` элементов управления и формы производится на этапе инициализации формы. При выборе элемента списка генерируется зависящая от состояния флажков строка, отображаемая в диалоговом окне. Причем, элемент первого столбца возвращается свойством `Text`, а второго и третьего — свойством `List`.

Изменение состояния флажка также будет вызывать выполнение процедуры обработки события `Click` списка при условии, что в нем имеются выбранные элементы. Последнее проверяется значением, возвращаемым свойством `ListIndex`. Дело в том, что это свойство возвращает индекс выбранного элемента. Если же такового нет, то оно возвращает значение −1.

Листинг 7.33. Скрытие данных в многостолбцовом списке

```
Private Sub UserForm_Initialize()
   ListBox1.ColumnCount = 3
   ListBox1.ColumnWidths = "80;0;0"
   Me.Caption = "Отдел кадров"
   ListBox1.RowSource = "Сотрудники!A2:C9"
   CheckBox1.Caption = "Должность"
   CheckBox2.Caption = "Стаж"
End Sub

Private Sub ListBox1_Click()
   Dim msg As String
   msg = "Фамилия: " & ListBox1.Text & vbCr
   If CheckBox1.Value Then
      msg = msg & "Должность: " & ListBox1.List(ListBox1.ListIndex, 1) _
          & vbCr
   End If
   If CheckBox2.Value Then
      msg = msg & "Стаж: " & ListBox1.List(ListBox1.ListIndex, 2) & vbCr
   End If
   MsgBox msg
End Sub

Private Sub CheckBox1_Change()
   If ListBox1.ListIndex >= 0 Then
      ListBox1_Click
   End If
End Sub
```

```
Private Sub CheckBox2_Change()
   If ListBox1.ListIndex >= 0 Then
      ListBox1_Click
   End If
End Sub
```

Вывод в многостолбцовом списке выбранного значения при помощи свойств *Text* и *Value*

Свойство `Text` по умолчанию возвращает выбранный элемент из первого столбца. Но эту установку можно изменить. Свойство `TextColumn` задает номер столбца, выбранный элемент из которого возвращается свойством `Text`. Кроме свойства `Text`, возвращающего выбранный элемент, в списке имеется свойство `Value`, действующее аналогично. Столбец, из которого оно возвращает элемент, задается свойством `BoundColumn`. При этом надо учитывать — как в свойстве `TextColumn`, так и в `BoundColumn` нумерация столбцов начинается с 1. Следующий код (листинг 7.34) является модификацией проекта предыдущего *разд. "Скрытие данных в многостолбцовом списке"*, только теперь данные из второго и третьего столбца считываются при помощи свойств `Text` и `Value`. Так как в списке три столбца (хотя последние два и скрыты), то значение из первого столбца считывается свойством `List`.

Листинг 7.34. Вывод в многостолбцовом списке выбранного значения при помощи свойств `Text` и `Value`

```
Private Sub UserForm_Initialize()
   ListBox1.ColumnCount = 3
   ListBox1.ColumnWidths = "80;0;0"
   Me.Caption = "Отдел кадров"
   ListBox1.RowSource = "Сотрудники!A2:C9"
   CheckBox1.Caption = "Должность"
   CheckBox2.Caption = "Стаж"
   ListBox1.TextColumn = 2
   ListBox1.BoundColumn = 3
End Sub

Private Sub ListBox1_Click()
   Dim msg As String
   msg = "Фамилия: " & ListBox1.List(ListBox1.ListIndex, 0) & vbCr
   If CheckBox1.Value Then
      msg = msg & "Должность: " & ListBox1.Text & vbCr
   End If
```

```
   If CheckBox2.Value Then
      msg = msg & "Стаж: " & ListBox1.Value & vbCr
   End If
   MsgBox msg
End Sub
```

Буксировка элементов из одного списка в другой

Объект `DataObject` может служить транспортером данных при программировании операции буксировки. Методы объекта `DataObject` позволяют контролировать процесс буксировки от ее начала до конца (табл. 7.18). Следующий код (листинг 7.35) демонстрирует подобное применение этого объекта при буксировке элементов из одного списка в другой.

Таблица 7.18. *Методы объекта* `DataObject`

Метод	Описание
`Clear`	Удаляет переносимые данные из объекта
`GetFormat`	Возвращает 1, если переносимые данные имеют строковый формат
`GetFromClipboard`	Копирует содержание буфера обмена в объект
`GetText`	Возвращает строку, переносимую в объекте
`PutInClipboard`	Перемещает данные из объекта в буфер обмена
`SetText`	Копирует строку в объект
`StartDrag`	Инициализирует операцию буксировки. Допустимыми возвращаемыми значениями являются следующие константы: `fmDropEffectNone`, `fmDropEffectCopy`, `fmDropEffectMove` и `fmDropEffectCopyOrMove`

При кодировании операции буксировки надо обрабатывать два события:

- `BeforeDragOver` — генерируется во время операции буксировки;
- `BeforeDropOrPaste` — генерируется непосредственно перед вставкой объекта.

Итак, для демонстрации того, как операция буксировки может быть запрограммирована, создайте форму, расположите в ней два списка и наберите следующий код (листинг 7.35). Проект готов.

Листинг 7.35. Буксировка элементов из одного списка в другой

```
Private sh As Integer
Private idx As Integer
```

```
Private Sub UserForm_Initialize()
   Dim i As Integer
   For i = 1 To 10
     ListBox1.AddItem ""Элемент " & i
   Next
End Sub

Private Sub ListBox1_MouseMove(ByVal Button As _
     Integer, ByVal Shift As Integer, ByVal X As _
     Single, ByVal Y As Single)
    Dim dobj As DataObject
    If Button = xlPrimaryButton Then
        Set dobj = New DataObject
        Dim Effect As Integer
        dobj.SetText ListBox1.Value
        Effect = dobj.StartDrag
        idx = ListBox1.ListIndex
        sh = Shift
    End If
End Sub

Private Sub ListBox2_BeforeDragOver(ByVal Cancel As _
    MSForms.ReturnBoolean, ByVal Data As _
    MSForms.DataObject, ByVal X As Single, _
    ByVal Y As Single, ByVal DragState As Long, _
    ByVal Effect As MSForms.ReturnEffect, _
    ByVal Shift As Integer)
    Cancel = True
    Effect = fmDropEffectCopyOrMove
End Sub

Private Sub ListBox2_BeforeDropOrPaste(ByVal _
    Cancel As MSForms.ReturnBoolean, _
    ByVal Action As Long, ByVal Data As _
    MSForms.DataObject, ByVal X As Single, _
    ByVal Y As Single, ByVal Effect As _
    MSForms.ReturnEffect, ByVal Shift As Integer)
    Cancel = True
    Effect = fmDropEffectCopyOrMove
    ListBox2.AddItem Data.GetText
    If sh = 2 Then
       ListBox1.RemoveItem idx
    End If
End Sub
```

Поле со списком (*ComboBox*)

Элемент управления **Поле со списком** (ComboBox) применяется для хранения списка значений. Он сочетает в себе функциональные возможности списка и поля ввода. В отличие от списка, в **Поле со списком** отображается только один элемент списка. У него отсутствует режим выделения нескольких элементов списка. Элемент управления **Поле со списком** позволяет пользователю вводить значение через поле ввода, как это делает элемент управления **Поле ввода** (TextBox). Свойства элемента управления **Поле со списком**, такие как `ListIndex`, `ListCount`, `List`, и методы `Clear`, `RemoveItem` и `AddItem` аналогичны соответствующим свойствам и методам элемента управления **Список**. Кроме того, у него есть ряд уникальных свойств, которые перечислены в табл. 7.19.

Таблица 7.19. *Свойства поля со списком*

Свойство	Описание
`DropButtonStyle`	Устанавливает вид поля со списком. Допустимыми значениями являются следующие константы: `fmDropButtonStylePlain` (кнопка без символов), `fmDropButtonStyleArrowDisplays` (кнопка со стрелкой), `fmDropButtonStyleEllipsis` (кнопка с эллипсом), `fmDropButtonStyleReduce` (кнопка с линией)
`ListRows`	Устанавливает число элементов, отображаемых в поле со списком
`MatchRequired`	Допустимые значения: `True` (в поле ввода нельзя ввести значения, отличные от перечисленных в списке, т. е. в поле со списком отключается функция поля ввода) и `False` (в противном случае)
`MatchFound`	Допустимые значения: `True` (среди элементов поля со списком имеется элемент, который совпадает с элементом, вводимым в поле ввода) и `False` (в противном случае)

Поле со списком, ввод данных в алфавитном порядке и объект *Collection*

Создадим простое приложение, которое при вводе данных, например фамилий, через поле ввода поля со списком автоматически упорядочивает их в алфавитном порядке. При этом вводимые данные сразу же после нажатия клавиши <Enter> отображаются в поле со списком в алфавитном порядке. При закрытии окна все данные из поля со списком вводятся в ячейки рабочего листа (рис. 7.25).

Для реализации данного приложения воспользуемся объектом `Collection`, который представляет собой удобное динамическое хранилище других объ-

ектов. Для работы с объектом Collection имеется только одно свойство Count, возвращающее число элементов, и три метода: Item (возвращающий элемент), Add (добавляющий новый элемент) и Remove (удаляющий элемент). Приводимый ниже код, с одной стороны, является хорошим примером, демонстрирующим работу с объектом Collection (место в семействе, куда вставляется новый элемент, определяется при помощи алгоритма бинарного поиска), а с другой стороны, показывающим, как из поля ввода поля со списком вводятся данные в сам список.

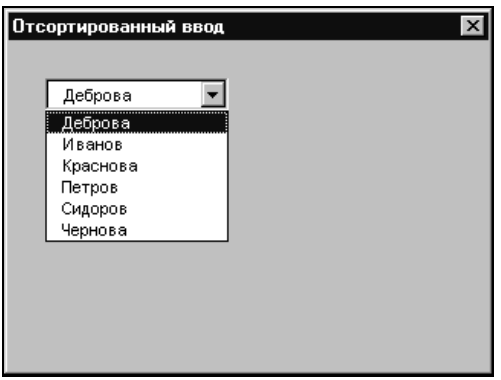


Рис. 7.25. Окно **Отсортированный ввод**

Итак, создайте форму, разместите на ней поле со списком, а в модуле формы наберите следующий код (листинг 7.36).

Листинг 7.36. Поле со списком и ввод данных в алфавитном порядке

```
Private ABC As Collection

Private Sub UserForm_Initialize()
   Me.Caption = "Отсортированный ввод"
   Set ABC = New Collection
End Sub

Private Sub ComboBox1_KeyDown(ByVal KeyCode As MSForms.ReturnInteger, _
                              ByVal Shift As Integer)
   If KeyCode = vbKeyReturn Then
      Dim i As Integer
      Insert ComboBox1.Text
      ComboBox1.Clear
      For i = 1 To ABC.Count
         ComboBox1.AddItem ABC.Item(i)
```

```
      Next
   End If
End Sub

Private Sub UserForm_Terminate()
   Range("A:A").Clear
   Dim i As Integer
   With ABC
      For i = 1 To .Count
        Cells(i, 1).Value = .Item(i)
      Next
   End With
   Columns(1).AutoFit
End Sub

Sub Insert(element As Variant)
   Dim min As Integer, max As Integer, middle As Integer
   If ABC.Count = 0 Then
      ABC.Add element
      Exit Sub
   End If

   min = 1
   max = ABC.Count

   If element <= ABC.Item(min) Then
      ABC.Add element, Before:=min
      Exit Sub
   End If

   If element >= ABC.Item(max) Then
      ABC.Add element, After:=max
      Exit Sub
   End If

   Do Until max – min <= 1
      middle = (min + max) / 2
      If ABC.Item(middle) < element Then
         min = middle
      Else
         max = middle
      End If
   Loop

   ABC.Add element, Before:=max
End Sub
```

Добавление и удаление данных в поле со списком

Рассмотрим простое приложение, в котором через поле ввода поля со списком возможно добавление новых элементов, *отличных* от элементов, уже имеющихся в списке. Таким образом, не допускается ввод элемента, совпадающего с одним из элементов списка. Этим рассматриваемый способ ввода данных отличается от того способа, который был рассмотрен в предыдущем разделе. Кроме того, из списка допускается удаление выбранных элементов и очистка всего списка. Первоначально список заполняется данными из диапазона ячеек. Если через поле ввода списка будет введен новый элемент, то он также вводится в диапазон, на основе которого заполнялся список. При удалении элемента из списка он удаляется и из диапазона. В обоих случаях у этого диапазона изменяются размеры, т. к. та или иная строка либо добавляется, либо удаляется. Поэтому приходится постоянно переопределять диапазон, на основе которого заполняется список.

Перейдем к конструированию приложения. Создайте форму с полем со списком и тремя кнопками. Установите значения свойства `Name` формы и элементов управления, используя окно **Properties**, как показано в табл. 7.20.

Таблица 7.20. *Значения свойств, установленные в окне* ***Properties***

Элемент управления	Значение свойства `Name`
Список	`cboNames`
Кнопка	`cmdAdd`
Кнопка	`cmdDelete`
Кнопка	`cmdClear`

Список будет заполняться данными, например фамилиями, из диапазона, расположенного в одном столбце с верхней ячейкой **A1**. Нажатие кнопки **Добавить** вызывает добавление в список нового элемента из поля ввода списка, нажатие кнопки **Удалить** приводит к удалению выбранного элемента из списка, а нажатие кнопки **Очистить** инициирует очистку списка (рис. 7.26).

Листинг 7.37. Добавление и удаление данных в поле со списком

```
Private Sub UserForm_Initialize()
   Me.Caption = "Заполнение списка без повторений"
   cmdAdd.Caption = "Добавить"
   cmdDelete.Caption = "Удалить"
   cmdClear.Caption = "Очистить"
```

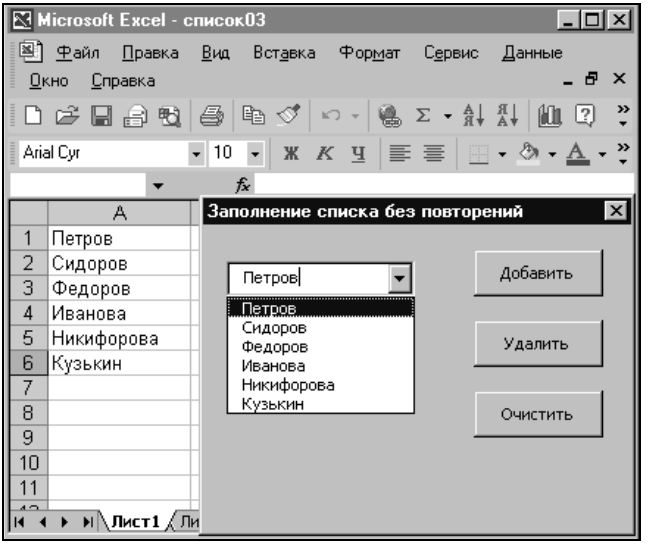


Рис. 7.26. Заполнение списка без повторений

```
   cboNames.RowSource = Range("A1").CurrentRegion.Address
   cboNames.ListIndex = 0
End Sub

Private Sub cmdAdd_Click()
   Dim el As String
   Dim n As Integer
   el = cboNames.Text
   If el = Empty Then Exit Sub
   If Range("A1").Value = Empty Then
        Range("A1").Value = el
   ElseIf Not cboNames.MatchFound Then
        n = Range("a1").CurrentRegion.Rows.Count
        Cells(n + 1, 1).Value = el
   Else
        MsgBox cboNames.Text & " уже есть в списке"
        Exit Sub
   End If
   cboNames.RowSource = Range("A1").CurrentRegion.Address
End Sub

Private Sub cmdDelete_Click()
   Dim n As Integer
   n = cboNames.ListIndex
```

```
   If n = -1 Or (n = 0 And cboNames.Text = Empty) Then
      Exit Sub
   End If
   Cells(n + 1, 1).EntireRow.Delete
   cboNames.RowSource = Range("A1").CurrentRegion.Address
End Sub

Private Sub cmdClear_Click()
   Range("A1").CurrentRegion.Clear
   cboNames.Text = Empty
End Sub
```

Рисунок (*Image*)

Элемент управления **Рисунок** (Image) используется для отображения графических файлов в формате BMP, CUR, GIF, ICO, JPG и WMF. В табл. 7.21 перечислены основные свойства элемента управления **Рисунок**.

Таблица 7.21. *Свойства элемента управления* ***Рисунок***

Свойство	Описание
`AutoSize`	Принимает логические значения и устанавливает, должен ли объект автоматически изменять размер для того, чтобы поместить изображение целиком
`Picture`	Задает отображаемый графический файл. Используется с функцией `LoadPicture`
`PictureSizeMode`	Устанавливает масштабирование рисунка. Допустимые значения: `fmPictureSizeModeClip` (не помещающиеся в границах объекта части рисунка обрезаются), `fmPictureSizeModeStretch` (рисунок масштабируется так, чтобы он занимал всю поверхность объекта), `fmPictureSizeModeZoom` (рисунок масштабируется с сохранением относительных размеров так, чтобы он помещался целиком внутри объекта)
`PictureAlignment`	Устанавливает расположение изображения внутри элемента управления. Допустимые значения: `fmPictureAlignmentTopLeft` (в верхнем левом углу), `fmPictureAlignmentTopRight` (в верхнем правом углу), `fmPictureAlignmentCenter` (в центре), `fmPictureAlignmentBottomLeft` (в нижнем левом углу), `fmPictureAlignmentBottomRight` (в нижнем правом углу)
`PictureTiling`	Принимает логические значения и устанавливает, надо ли покрывать объект мозаикой из рисунка

Окно *О программе*

Рисунок часто используется при создании окон типа **О программе** или **Об авторе** для вставки в окно растровых изображений. Следующий проект является демонстрацией подобного применения рисунка (рис. 7.27). Для его реализации создайте окно, в котором расположите рисунок и две надписи. Кроме того, потребуется растровый файл (в данном случае Me.bmp) с вашим или каким-либо другим изображением. Этот файл надо поместить в тот же каталог, где находится и XLS-файл с вашей книгой. В модуле формы наберите приводимый ниже код. Обратите внимание на то, что в различных надписях для придания окну большей презентабельности используются различные установки параметров шрифта.



Рис. 7.27. Окно **О программе**

Листинг 7.38. Окно О программе

```
Private Sub UserForm_Initialize()
   Me.Caption = "О программе"
   Image1.BorderStyle = fmBorderStyleNone
   Image1.Picture = LoadPicture("me.bmp")
   Image1.PictureSizeMode = fmPictureSizeModeZoom
   Label1.Font.Size = 12
   Label1.Font.Bold = True
   Label1.Caption = "Учебный пример"
   Label2.Font.Size = 11
   Label2.Font.Italic = True
   Label2.Caption = "Андрей Гарнаев"
End Sub
```

Просмотр слайдов

Рисунок позволяет создавать простое средство для просмотра слайдов. Как это делается, продемонстрируем на простом примере. Здесь создается приложение, позволяющее просматривать, как выглядят книги автора, относящиеся к VBA, т. е. к той тематике, которой и посвящена данная книга (рис. 7.28). Итак, создайте форму, в которой расположите рисунок и список. Кроме того, потребуются файлы с соответствующими растровыми изображениями (в данном случае 1.gif, 2.gif, 3.gif), которые следует расположить в том же каталоге, где находится и XLS-файл с вашей книгой. В модуле формы наберите приводимый ниже код (листинг 7.39). Вот и все. Проект готов.

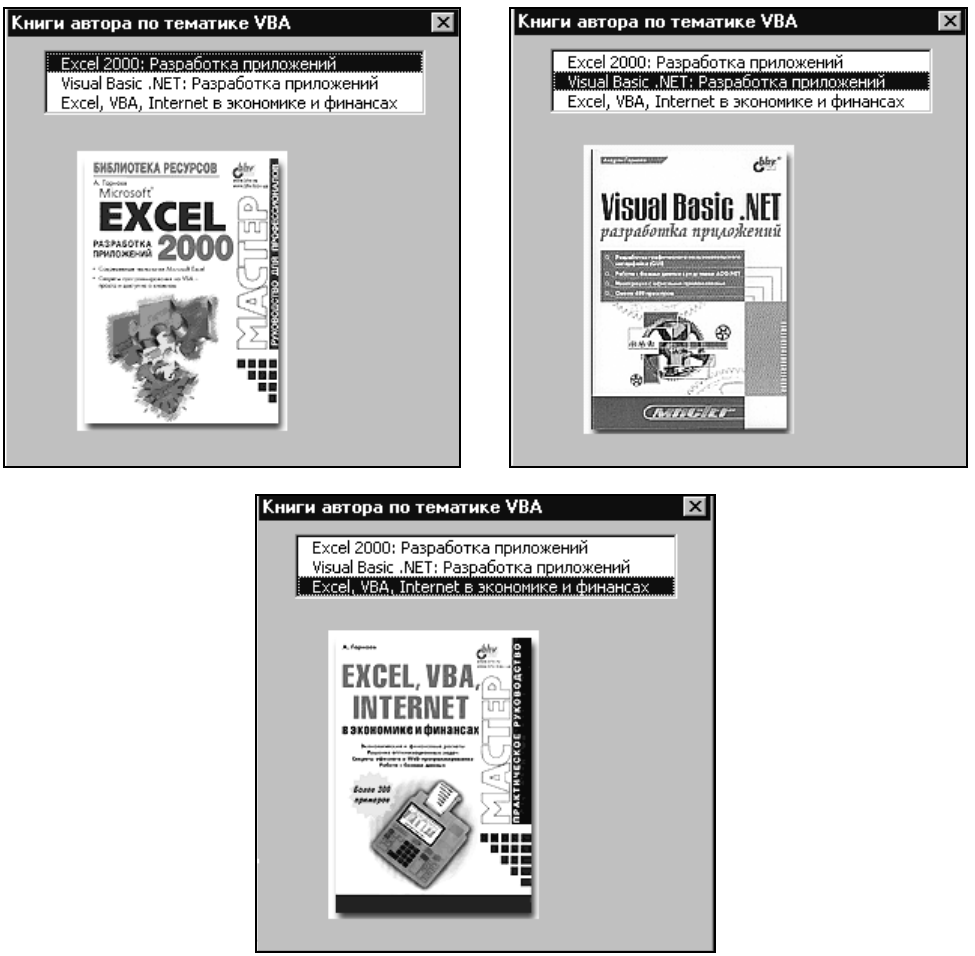


Рис. 7.28. Просмотр слайдов

Листинг 7.39. Просмотр слайдов

```
Private Sub UserForm_Initialize()
   Me.Caption = "Книги автора по тематике VBA"
   Image1.BorderStyle = fmBorderStyleNone
   ListBox1.AddItem "Excel 2000: Разработка приложений"
   ListBox1.AddItem "Visual Basic .NET: Разработка приложений"
   ListBox1.AddItem "Excel, VBA, Internet в экономике и финансах"
   ListBox1.ListIndex = 0
End Sub

Private Sub ListBox1_Click()
   Dim bs As Variant
   bs = Array("1.gif", "2.gif", "3.gif")
   Image1.Picture = LoadPicture(bs(ListBox1.ListIndex))
End Sub
```

Модифицированный мастер диаграмм

Построение графика с помощью мастера диаграмм предполагает довольно большой объем подготовительной работы: заполнение диапазона значениями аргумента, заполнение другого диапазона значениями функции и непосредственное построение мастером диаграмм графика в четыре шага.

Попытаемся создать приложение, которое само заполняет диапазоны со значениями аргумента и функции, строит диаграмму и отображает диаграмму не только на рабочем листе, но и в форме. В этом приложении от пользователя будет требоваться только ввод границ интервала, на котором строится график, и самой функции, причем, функцию надо вводить не в виде формулы рабочего листа, а в привычном виде, в качестве аргумента которой нужно использовать символ "z" (рис. 7.29). Использование символа "z" в качестве аргумента функции связано с целью упрощения кода.

Перейдем к построению приложения. Создайте форму с тремя надписями, тремя полями ввода, рисунком и кнопкой. Установите значения свойства `Name` элементов управления в окне **Properties**, как показано в табл. 7.22.

Таблица 7.22. *Значения свойства* `Name` *элементов управления, установленные в окне* ***Properties***

Элемент управления	Значение свойства `Name`	Описание
Поле ввода	`txtBegin`	Ввод левого конца интервала, на котором строится график

Таблица 7.22 *(окончание)*

Элемент управления	Значение свойства Name	Описание
Надпись	lblBegin	Надпись, соответствующая полю ввода txtBegin
Поле ввода	txtEnd	Ввод правого конца интервала, на котором строится график
Надпись	lblEnd	Надпись, соответствующая полю ввода txtEnd
Поле ввода	txtFun	Ввод формулы функции, для которой строится график. Формула должна быть записана по правилам программирования, в качестве аргумента использован символ "z"
Надпись	lblFun	Надпись, соответствующая полю ввода txtFun
Рисунок	imgFun	Отображается растровое изображение из файла Graph.gif, который содержит графический образ диаграммы
Кнопка	cmdReady	График функции строится по 101 точке. Первоначально определяется шаг табуляции аргумента функции как результат деления разности между начальным и конечным значениями аргумента на 100. После этого начальное значение вводится в ячейку **A2**, и при помощи метода DataSeries производится табуляция аргумента функции вниз по столбцу. В формуле функции заменяется аргумент z ссылкой на ячейку **A2**, а перед формулой помещается знак равенства. Полученная таким образом формула рабочего листа вводится в ячейку **B2**. Методом AutoFill формула буксируется вниз по столбцу так, чтобы найти значения функции для всех протабулированных значений аргумента. По протабулированным значениям аргумента и функции строится диаграмма, графический образ которой экспортируется в файл Graph.gif. Отображается растровое изображение из файла Graph.gif в рисунке imgFun.

Для завершения создания приложения в модуле формы наберите приводимый ниже код (листинг 7.40). В качестве параметра функции используется символ "z", т. к. данный символ не входит в имена встроенных функций рабочего листа. Это позволяет, используя только одну инструкцию

```
f = Replace(LCase(f), "z", "A2")
```

заменить все вхождения параметра в функцию ссылкой на ячейку **A2**. Если бы в качестве аргумента использовался символ "x", то здесь в общем случае не обойтись одной инструкцией. Дело в том, что символ "x" входит в имя

функции `Exp`. Поэтому применение данной инструкции будет заменять ссылкой на ячейку **A2** не только аргумент, но и встречаемые упоминания имени функции `Exp` на `EA2p`, что, конечно, приведет к генерации ошибки.

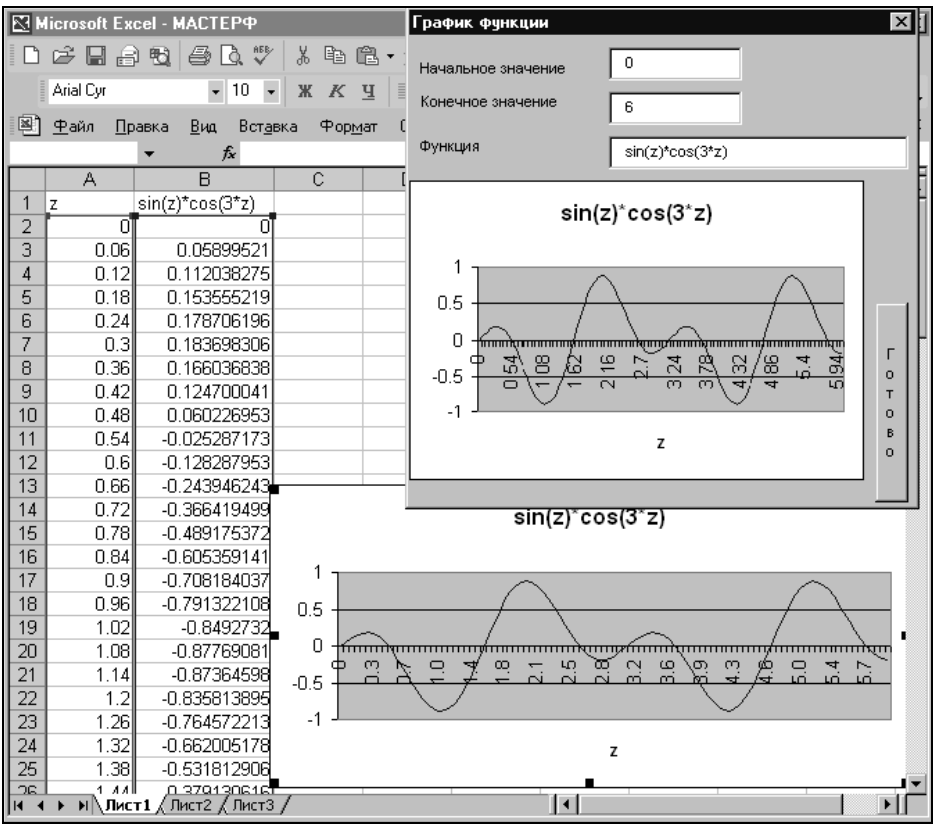


Рис. 7.29. Построение графика модифицированным мастером диаграмм

Листинг 7.40. Модифицированный мастер диаграмм

```
Private Sub UserForm_Initialize()
   Me.Caption = "График функции"
   lblBegin.Caption = "Начальное значение"
   lblEnd.Caption = "Конечное значение"
   lblFun.Caption = "Функция"
   With cmdReady
      .Caption = "Готово"
      .WordWrap = True
   End With
```

```
   With imgFun
      .BorderStyle = fmBorderStyleNone
      .PictureAlignment = fmPictureAlignmentTopLeft
      .PictureSizeMode = fmPictureSizeModeZoom
   End With
End Sub

Private Sub cmdReady_Click()
   Dim a As Double
   Dim b As Double
   Dim h As Double
   Dim f As String

   Range("A:B").Clear
   If ActiveSheet.ChartObjects.Count > 0 Then
      ActiveSheet.ChartObjects.Delete
   End If
   a = CDbl(txtBegin.Text)
   b = CDbl(txtEnd.Text)
   h = (b - a) / 100
   f = Trim(txtFun.Text)
   f = Replace(LCase(f), "z", "A2")
   f = "=" & f
   Range("A2").Value = a
   Range("A2").DataSeries Rowcol:=xlColumns, _
                                 Step:=h, _
                                 Stop:=b
   Dim rgnX As Range
   Dim rgnY As Range
   Set rgnX = Range("A2").CurrentRegion
   Set rgnY = rgnX.Offset(0, 1)
   Range("B2").Value = f
   Range("B2").AutoFill Destination:=Range(rgnY.Address)
   Range("A1").Value = "z"
   Range("B1").Value = Trim(txtFun.Text)
   Dim shname As String
   shname = ActiveSheet.Name
   Charts.Add
   With ActiveChart
      .ChartType = xlLine
      .SetSourceData Source:=Sheets(shname).Range(rgnY.Address), _
                     PlotBy:=xlColumns
      .SeriesCollection(1).XValues = Sheets(shname).Range(rgnX.Address)
      .Location Where:=xlLocationAsObject, Name:=shname
   End With
```

```
   With ActiveChart
      .HasTitle = True
      .ChartTitle.Characters.Text = Trim(txtFun.Text)
      .Axes(xlCategory, xlPrimary).HasTitle = True
      .Axes(xlCategory, xlPrimary).AxisTitle.Characters.Text = "z"
      .HasLegend = False
      .Export Filename:="graph.gif", FilterName:="GIF"
   End With
   imgFun.Picture = LoadPicture("graph.gif")
End Sub
```

Элемент управления *RefEdit*

Элемент управления **RefEdit** аналогичен полю ввода, но позволяет вводить в него ссылку на диапазон выбором на рабочем листе. Свойство `Value` возвращает эту ссылку.

Определение статистических параметров диапазона

В качестве примера использования элемента управления **RefEdit** сконструируем простой проект, определяющий некоторые статистические параметры диапазона, а именно — максимальное, минимальное значения, а также сумму всех значений ячеек этого диапазона. Итак, создайте форму, на которой расположите кнопку и элемент управления **RefEdit** (рис. 7.30). В модуле формы наберите следующий код (листинг 7.41). Проект готов. Для вычисления максимального, минимального значений, а также суммы всех значений ячеек диапазона используются свойства `Min`, `Max` и `Sum` объекта `WorksheetFunction`, представляющие собой одноименные функции рабочего листа.

Листинг 7.41. Статистика

```
Private Sub UserForm_Initialize()
   Me.Caption = "Статистика"
   CommandButton1.Caption = "OK"
End Sub

Private Sub CommandButton1_Click()
   Dim r As String
   Dim min As Double
   Dim max As Double
   Dim s As Double
   r = RefEdit1.Value
   Dim rgn As Range
   Set rgn = Range(r)
```

```
    min = WorksheetFunction.Min(rgn)
    max = WorksheetFunction.Max(rgn)
    s = WorksheetFunction.Sum(rgn)
    MsgBox RefEdit1.Value & vbCr & _
           "min=" & min & vbCr & _
           "max=" & max & vbCr & _
           "s=" & s
End Sub
```

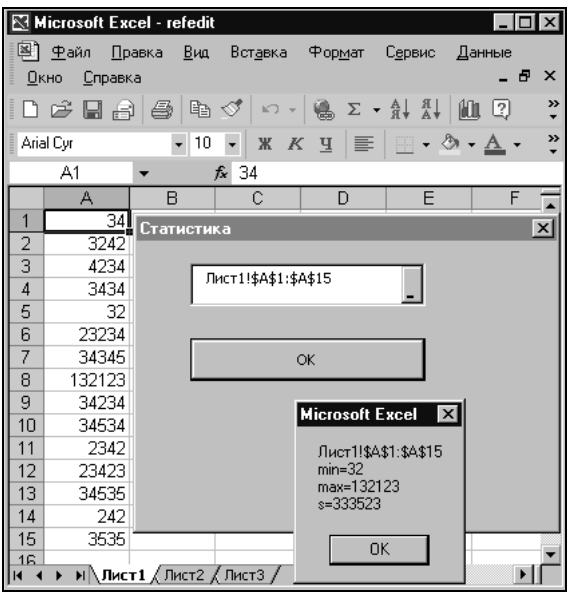


Рис. 7.30. Статистика

Решение системы линейных уравнений

В качестве еще одного примера использования элемента управления **RefEdit** создадим проект по решению систем линейных уравнений $AX = B$ (листинг 7.42). Здесь A — квадратная $n \times n$ матрица, B — столбец свободных членов, а X — столбец неизвестных. При решении системы уравнений используются функции рабочего листа МОБР и МУМНОЖ, возвращающие матрицу, обратную к данной, а также результат перемножения двух матриц. Кроме того, для проверки существования решения системы применяется функция МОПР, возвращающая определитель квадратной матрицы.

В окне формы имеется элемент управления **RefEdit** и кнопка. Пользователь вводит в элемент управления **RefEdit** ссылку на диапазон размера $n \times (n+1)$, первые n столбцов которого содержат матрицу коэффициентов, а последний

столбец — столбец свободных членов. Нажатие на кнопку вызывает вычисление решения системы и вывод его в смежный справа с выделенным диапазон.

Например, на рис. 7.31 решена следующая система линейных уравнений:

$$12\ x_1 + 3\ x_2 = 2;$$

$$23\ x_1 + 45\ x_2 = 3.$$

Матрица коэффициентов расположена в диапазоне **B3:C4**, столбец свободных членов — в диапазоне **D3:D4**, решение системы выводится в диапазон **E3:E4**.

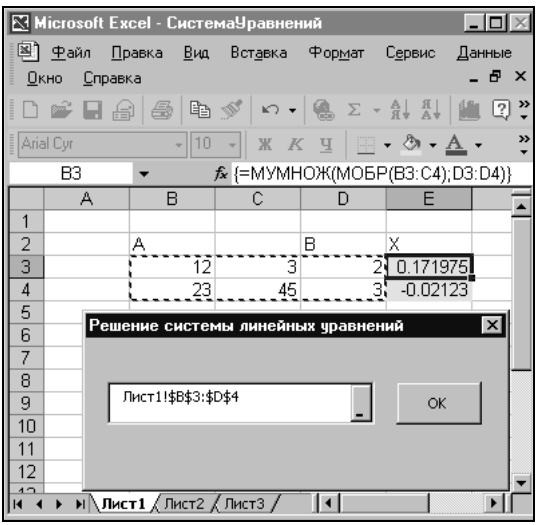


Рис. 7.31. Решение системы линейных уравнений

Листинг 7.42. Решение системы линейных уравнений

```
Private Sub cmdOK_Click()
   Dim s As String
   s = RefEdit1.Text
   Dim r As Range
   Dim rA As Range
   Dim rB As Range
   Dim rX As Range
   Set r = Range(s)
   Set rB = r.Columns(r.Columns.Count)
   Set rA = r.Resize(ColumnSize:=r.Columns.Count - 1)
   Set rX = rB.Offset(ColumnOffset:=1)
```

```
   Dim adA As String
   Dim adB As String
   adA = rA.Address(ReferenceStyle:=xlR1C1, _
                    RowAbsolute:=False, _
                    ColumnAbsolute:=False, _
                    RelativeTo:=rX)
   adB = rB.Address(ReferenceStyle:=xlR1C1, _
                    RowAbsolute:=False, _
                    ColumnAbsolute:=False, _
                    RelativeTo:=rX)
   Dim d As Double
   d = Application.Evaluate("=MDETERM(" & rA.Address & ")")
   If d = 0 Then
      MsgBox "Система не имеет решения"
   End If
   rX.FormulaArray = "=MMULT(MINVERSE(" & adA & ")," & adB & ")"
End Sub
```

Набор страниц (*MultiPage*) и набор вкладок (*TabStrip*)

Элемент управления **Набор страниц** (MultiPage) позволяет реализовать многостраничные диалоговые окна. Заголовки страниц отображаются на вкладках. Переход от страницы к странице осуществляется щелчком на вкладке. Создать, переименовать, удалить или переместить страницу элемента управления **Набор страниц** можно вручную, выбрав ярлык соответствующего листа и вызвав щелчком правой кнопки мыши контекстное меню. Объект `MultiPage` содержит в себе семейство `Pages`, являющееся набором всех страниц, входящих в этот объект. В табл. 7.23 перечислены основные свойства объекта `MultiPage`.

Таблица 7.23. *Основные свойства объекта* `MultiPage`

Свойство	Описание
`Value`	Возвращает или устанавливает номер активной страницы. Нумерация производится с нуля
`MultiRow`	Свойство, принимающее логические значения. Если его значение равно `True`, то ярлыки, не помещающиеся в одну строчку, выводятся в несколько строчек. Если его значение равно `False`, то в случае, когда ярлыки не помещаются в одну строчку, появляется полоса прокрутки, позволяющая переходить от страницы к странице
`SelectedItem`	Возвращает выбранную страницу

Семейство `Pages` состоит из всех объектов `Page`, т. е. страниц объекта `MultiPage`. Семейство `Pages` имеет единственное свойство `Count`, возвращающее число элементов семейства. Методы семейства `Pages` перечислены в табл. 7.24.

Таблица 7.24. *Методы семейства* `Pages`

Метод	Описание
`Add`	Создает новую страницу
`Clear`	Удаляет все страницы из семейства `Pages`
`Remove`	Удаляет страницу из семейства `Pages`
`Item`	Возвращает страницу с указанным в качестве индексом

Элемент управления **Набор вкладок** (TabStrip) создает несколько вкладок в диалоговом окне и по своим функциональным возможностям эквивалентен набору страниц. Объект `TabStrip` содержит в себе семейство `Tabs`, представляющее собой набор всех вкладок. Объект `TabStrip` и семейство `Tabs` обладают теми же свойствами и методами, что и объект `MultiPage` и семейство `Pages`.

Статистика и набор страниц

В качестве примера приложения с набором страниц улучшим интерфейс проекта из вышеприведенного *разд. "Определение статистических параметров диапазона"* (рис. 7.32). Создайте форму, на которой расположите набор страниц. На первой странице, значение `Caption` которой установите равным `Вычисления`, разместите элемент управления **RefEdit**, поле ввода и кнопку. На второй странице, значение `Caption` которой установите равным `Параметры`, разместите три флажка.

Используя окно **Properties**, установите значения свойств элементов управления, как показано в табл. 7.25.

Таблица 7.25. *Значения свойств, установленные в окне* ***Properties***

Объект	Свойство	Значение
Поле ввода	`Name`	`txtStat`
Кнопка	`Name`	`cmdOK`
	`Caption`	`OK`
Флажок	`Name`	`chkSum`
	`Caption`	`Сумма`

Таблица 7.25 *(окончание)*

Объект	Свойство	Значение
Флажок	Name	chkMax
	Caption	Максимум
Флажок	Name	chkMin
	Caption	Минимум

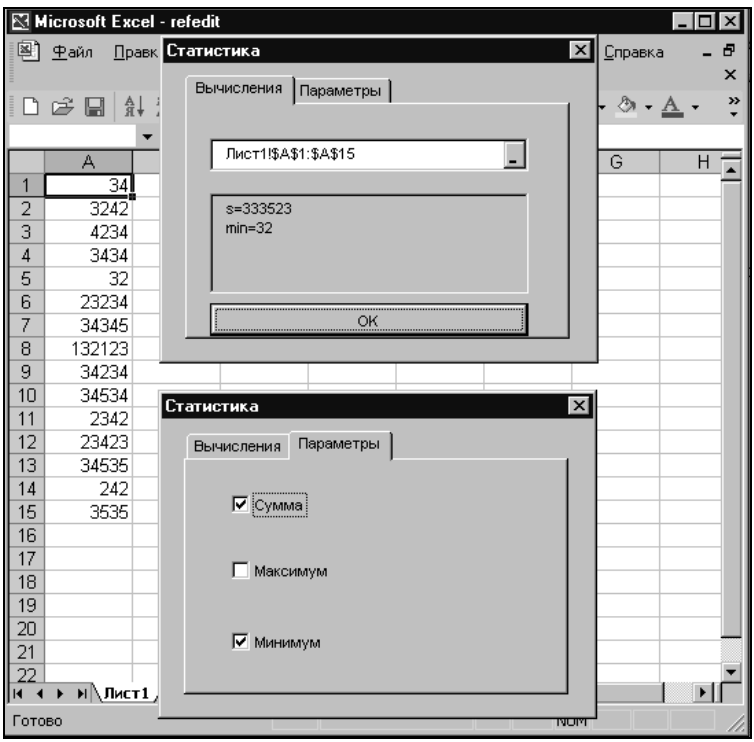


Рис. 7.32. Статистические параметры диапазона и набор страниц

В модуле формы наберите код, представленный в листинге 7.43. Проект готов. В поле ввода выводится в несколько строчек интегрированная статистика, параметры которой устанавливаются флажками страницы **Параметры**. Поле ввода заблокировано для корректировки результатов со стороны пользователя. Кроме того, цвет фона поля установлен равным цвету фона формы. Поэтому визуально поле ввода воспринимается как вдавленная в форму надпись.

Листинг 7.43. Статистика и набор страниц

```
Private Sub UserForm_Initialize()
   txtStat.Locked = True
   txtStat.MultiLine = True
   txtStat.BackColor = BackColor
End Sub

Private Sub cmdOK_Click()
   Dim r As String
   Dim m As Double
   Dim max As Double
   Dim s As Double
   r = RefEdit1.Value
   Dim rgn As Range
   Set rgn = Range(r)
   Dim msg As String
   If chkSum.Value Then
      s = WorksheetFunction.Sum(rgn)
      msg = "s=" & s & vbCr
   End If
   If chkMax.Value Then
      max = WorksheetFunction.max(rgn)
      msg = msg & "max=" & max & vbCr
   End If
   If chkMin.Value Then
      min = WorksheetFunction.min(rgn)
      msg = msg & "min=" & min & vbCr
   End If
   If msg = Empty Then
      txtStat.Text = "Nothing"
   Else
      txtStat.Text = msg
   End If
End Sub
```

Последовательность перехода элементов управления

Последовательность перехода определяет порядок, в котором получают фокус элементы управления при нажатии клавиши <Tab>. При нажатии комбинации клавиш <Shift>+<Tab> элементы управления получают фокус

в обратном порядке. Для установки последовательности перехода формы необходимо:

1. Находясь в Редакторе Visual Basic, выбрать команду **View | Tab Order**.
2. В появившемся диалоговом окне **Последовательность перехода** с помощью кнопок **Вниз** и **Вверх** изменить последовательность перехода выделенного элемента управления в зависимости от потребности.

Другим способом задания последовательности перехода является определение значения свойства `TabIndex` элемента управления. При этом надо помнить, что начальному элементу соответствует значение свойства `TabIndex`, равное 0, второму — 1, третьему — 2 и т. д. Если значение свойства `TabStop` равно `False`, то элемент управления не получает фокус при обходе элементов управления нажатием клавиши <Tab>. Если же значение свойства `TabStop` равно `True` (значение, используемое по умолчанию), то элемент управления получает фокус.

Отображение встроенных диалоговых окон

VBA позволяет программно отображать на экране встроенные в MS Excel диалоговые окна наряду с пользовательскими диалоговыми окнами. Все встроенные диалоговые окна в MS Excel образуют семейство `Dialogs`, параметр которого специфицирует активизируемое диалоговое окно (табл. 7.26). Отображение встроенного диалогового окна на экране осуществляется методом `Show`. Например, следующая процедура (листинг 7.44) при щелчке на кнопке активизирует диалоговое окно **Открытие документа** (`Open`) (рис. 7.33).

Листинг 7.44. Отображение диалогового окна Открытие документа

```
Private Sub CommandButton1_Click()
   Application.Dialogs(xlDialogOpen).Show
End Sub
```

Таблица 7.26. *Значения параметра семейства* `Dialogs`

Значение параметра	Диалоговое окно
`xlDialogFindFile`	Открытие документа при поиске файла
`xlDialogFileDelete`	Удалить файл
`xlDialogGoalSeek`	Подбор параметра
`xlDialogSaveAs`	Сохранить как
`xlDialogSaveWorkbook`	Сохранить

Таблица 7.26 *(окончание)*

Значение параметра	Диалоговое окно
`xlDialogPrint`	Печать
`xlDialogPrintPreview`	Предварительный просмотр

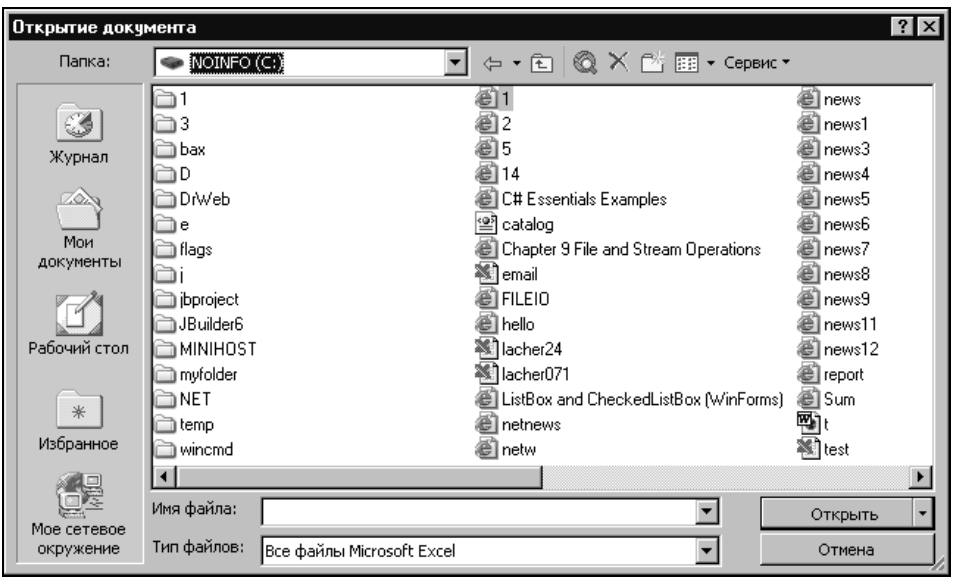


Рис. 7.33. Диалоговое окно **Открытие документа**

В методе `Show` можно указывать значения параметров, управляющих выводом в диалоговом окне специфицированной информации. Например, при отображении диалогового окна **Подбор параметра** можно указать значения параметров `target_cell` (поле ввода **Установить в ячейке**), `target_value` (поле ввода **Значение**), `variable_cell` (поле ввода **Изменяя значение ячейки**), в результате чего на экране отобразится окно с заполненными полями ввода (листинг 7.45).

Листинг 7.45. Отображение диалогового окна Подбор параметра с введенными значениями

```
Private Sub DoGoalSeek ()
  Dim fl As Boolean
  fl = Application.Dialogs(xlDialogGoalSeek).Show(Range("A1"), _
                                                  0, Range("A2"))
```

```
  If fl Then
     MsgBox "Решение найдено"
  Else
     MsgBox "Решение не найдено"
  End If
End Sub
```

Открытие документа и метод *GetOpenFilename*

Существует еще один способ отображения окна **Открытие документа**, а именно методом `GetOpenFilename` объекта `Application`. Этот метод приводит к отображению на экране окна **Открытие документа**, но не к открытию выбранного при помощи этого окна документа. Метод просто возвращает имя выбранного файла или значение `False`, если файл не был выбран. Для открытия же выбранного файла надо дополнительно воспользоваться методом `Open` семейства `Workbooks` (листинг 7.46).

Листинг 7.46. Открытие файла

```
Sub OpenDoc()
   Dim FName As Variant
   FName = Application.GetOpenFilename( _
      "Книга Microsoft Excel (*.xls), *.xls")
   If FName <> False Then
      Workbooks.Open FName
   Else
      MsgBox "Файл не выбран"
   End If
End Sub
```

Устанавливая значения параметров метода `GetOpenFilename`, можно управлять видом диалогового окна. В общем случае этот метод имеет следующий синтаксис:

```
GetOpenFilename(FileFilter, FilterIndex, Title, ButtonText, MultiSelect)
```

- *`FileFilter`* — необязательный парамер, являющийся строкой, задающей фильтр для отображаемых файлов. Если этот параметр опущен, то его значение по умолчанию полагается равным `"Все файлы (*.*),*.*"`. Еще одним примером фильтра может быть, например, строка `"Книга Microsoft Excel (*.xls), *.xls, Растровые файлы (*.bmp), *.bmp"` (Здесь через запятую указываются те типы файлов, которые будут выводиться в списке **Тип файлов**);
- *`FilterIndex`* — необязательный параметр, задающий индекс фильтра из списка **Тип файлов**, который используется по умолчанию;

- *Title* — необязательный параметр, определяющий заголовок окна;
- *ButtonText* — используется только в Macintosh;
- *MultiSelect* — необязательный параметр, принимающий логические значения. Если его значение равно True, то допустим выбор нескольких файлов. В последнем случае метод возвращает не строку, а массив строк.

Так как метод GetOpenFilename не совершает действий над файлами, а только возвращает имена выбранных файлов, то отображаемое окно можно использовать для функций, отличных от открытия выбранного файла — например, удаления выбранных файлов (листинг 7.47). При этом, конечно, стоит изменить заголовок окна в соответствии с его новыми функциями.

Листинг 7.47. Удаление файлов

```
Sub DeleteFile()
   Dim FName As Variant
   FName = Application.GetOpenFilename( _
               FileFilter:="Книга Microsoft Excel (*.xls), *.xls", _
               MultiSelect:=True, Title:="Удаление файла")
   If Not IsArray(FName) Then
      MsgBox "nothing"
      Exit Sub
   End If
   Dim i As Integer
   For i = LBound(FName) To UBound(FName)
       Kill FName(i)
   Next
End Sub
```

Простейший браузер для графических файлов

Метод GetOpenFilename возвращает имя выбранного файла любой природы. В частности, он может вернуть и растровый файл. С другой стороны, в рисунок может быть загружено изображение любого растрового файла. Вместе эти два факта обеспечивают возможность построения простого средства просмотра растровых файлов (рис. 7.34). Итак, создайте форму, расположите в ней кнопку и рисунок, а в модуле формы наберите следующий код (листинг 7.48). Проект готов.

Листинг 7.48. Простейший браузер для графических файлов

```
Private Sub UserForm_Initialize()
   Me.Caption = "Просмотр bmp и gif-файлов"
```

```
   CommandButton1.Caption = "Открыть"
End Sub

Private Sub CommandButton1_Click()
   OpenDoc
End Sub

Sub OpenDoc()
   Dim FName As Variant
   FName = Application.GetOpenFilename( _
      FileFilter:="gif файлы(*.gif), *.gif, bmp файлы (*.bmp), *.bmp", _
      FilterIndex:=2)
   If FName <> False Then
        Image1.Picture = LoadPicture(FName)
   Else
       MsgBox "Файл не выбран"
   End If
End Sub
```

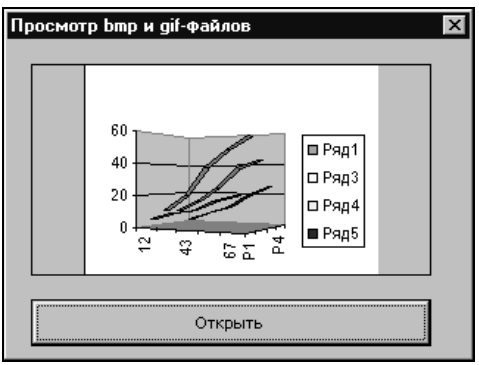


Рис. 7.34. Простейший браузер для графических файлов

Сохранение документа и метод *GetSaveAsFilename*

Метод `GetSaveAsFilename` объекта `Application` отображает на экране окно **Сохранение документа**. Этот метод так же не выполняет сохранение файла, а только возвращает имя сохраняемого файла, выбранного в этом окне. Сохранить же файл можно, дополнив код инструкцией, в которой применяется один из методов `SaveAs` или `Save` объекта `Workbook`.

Синтаксис метода `GetSaveAsFilename` похож на синтаксис метода `GetOpenFilename`.

```
GetSaveAsFilename(InitialFilename, FileFilter, FilterIndex,
↳Title, ButtonText)
```

- *InitialFilename* — необязательный параметр, задающий любое допустимое имя файла, которое будет появляться в поле **Имя файла** (по умолчанию, там появляется имя файла Книга);
- *FileFilter* — необязательный параметр, устанавливающий фильтр для отображаемых файлов. Если вы хотите, чтобы к имени файла автоматически добавлялось расширение, то использование этого параметра обязательно;
- *FilterIndex* — необязательный параметр, указывающий, какой фильтр из списка **Тип файлов** будет использоваться по умолчанию;
- *Title* — необязательный параметр, определяющий заголовок окна;
- *ButtonText* — используется только в Macintosh.

Например, следующий код (листинг 7.49) можно использовать при сохранении рабочей книги.

Листинг 7.49. Сохранение файла

```
Sub SaveDoc()
   Dim FName As Variant
   FName = Application.GetSaveAsFilename( _
           InitialFilename:="Отчет", _
           FileFilter:="Книга Microsoft Excel (*.xls), *.xls")
   If FName <> False Then Application.ThisWorkbook.SaveAs FName
End Sub
```

Дополнительные элементы управления

В VBA кроме перечисленных стандартных элементов управления имеется ряд дополнительных. Дополнительные элементы управления являются самостоятельными объектами, обладающими как общими для всех элементов управления свойствами и методами, так и присущими только им свойствами и методами.

Добавление дополнительного элемента управления

Добавляются элементы управления в панель элементов следующим образом:

1. Выберите команду **Tools | Additional Controls**.
2. В отобразившемся окне **Additional Controls** (рис. 7.35) в списке **Available Controls** установите флажок напротив добавляемого элемента управления.
3. Нажмите кнопку **OK**.

В результате значок выбранного дополнительного элемента управления появится в панели элементов **Toolbox**.

Примечание

Распространять приложение, имеющее дополнительный элемент управления, необходимо совместно с соответствующим OCX-файлом. Имя OCX-файла и его местоположение можно узнать из надписи **Location** окна **Additional Controls**.

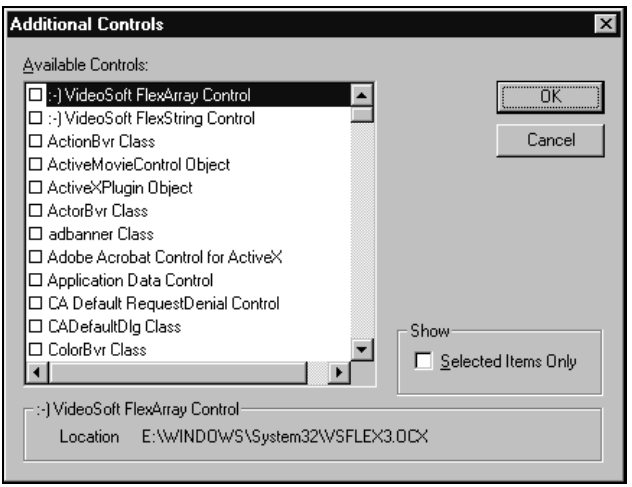


Рис. 7.35. Окно **Additional Controls**

Удаление дополнительного элемента управления

Удаляется ненужный элемент управления из панели элементов почти аналогично тому, как добавляется в нее, а именно:

1. Выберите команду **Tools | Additional Controls**.
2. В отобразившемся на экране окне **Additional Controls** в списке **Available Controls** снимите флажок напротив удаляемого элемента управления.
3. Нажмите кнопку **OK**.

Пример дополнительного элемента управления — *Календарь* (Calendar)

Среди дополнительных элементов управления очень полезным является элемент управления **Календарь** (Calendar). Этот элемент управления предоставляет средство для организации удобного интерфейса по вводу дат. Элемент управления конструируется на форме кнопкой **Calendar** . В табл. 7.27 и 7.28 перечислены основные свойства и методы элемента управления **Календарь**.

Таблица 7.27. *Свойства элемента управления* ***Календарь***

Свойство	Описание
`Day`	Возвращает выбранный день
`DayFont`, `DayFontColor`	Устанавливают шрифт и цвет шрифта для названий дней недели
`DayLenght`	Допустимые значения: 2 — отображаются полные названия дней, 1 — в русской версии отображаются две буквы из названий дней, в английской — три буквы, 0 — отображается только первая буква из названия дня
`FirstDay`	Первый день недели. Допустимые значения от воскресенья до субботы
`Month`	Возвращает выбранный месяц
`MonthLenght`	Допустимые значения: 2 — отображаются полные названия месяца, 0 — отображаются только первые три буквы из названия месяца
`ShowDays`	Допустимые значения: `True` (отображаются названия дней недели) и `False` (в противном случае)
`ShowDataSelected`	Допустимые значения: `True` (отображается выбранная дата в верхней части календаря) и `False` (в противном случае)
`Value`	Возвращает или устанавливает выбранную дату
`Year`	Возвращает или устанавливает выбранный год

Таблица 7.28. *Методы элемента управления* ***Календарь***

Метод	Описание
`NextDay`, `NextWeek`, `NextMonth`, `NextYear`	Устанавливают следующий день, неделю, месяц и год
`PreviousDay`, `PreviousWeek`, `PreviousMonth`, `PreviousYear`	Устанавливают предыдущий день, неделю, месяц и год
`Today`	Устанавливает текущую дату в календаре

Приложение *Заказы* и определение элемента управления, на котором был установлен фокус

Рассмотрим пример простого приложения, которое позволяет вводить при помощи элемента управления **Календарь** начальную и конечную дату выполнения заказа и составлять таблицу учета выполнения заказов сотрудни-

ками. Фамилии сотрудников перечисляются в поле со списком, отображаемым в окне. Проблема, которая возникает при составлении подобных приложений, — необходимость определения поля ввода (а их два — для начальной и конечной дат), в которое следует вводить дату по щелчку на календаре. Эта проблема решается анализом события `Enter`, которое возникает при получении элементом управления фокуса. В переменную уровня модуля в процедурах обработки события `Enter` полей ввода будет вводиться число, идентифицирующее поле ввода. Поэтому текущее значение этой переменной и будет определять последнее поле ввода, получившее фокус. Итак, создайте форму, на ней расположите три надписи, поле со списком, два поля ввода, календарь и кнопку. Присвойте значения свойства `Name` поля со списком, полей ввода, кнопки и календаря равными: `cmbName`, `txtEnd`, `txtBegin`, `cmdOK`, `cldMy`. Установите значения свойства `Caption` формы, кнопки и надписей, как на рис. 7.36. В модуле формы наберите следующий код (листинг 7.50). Проект готов.

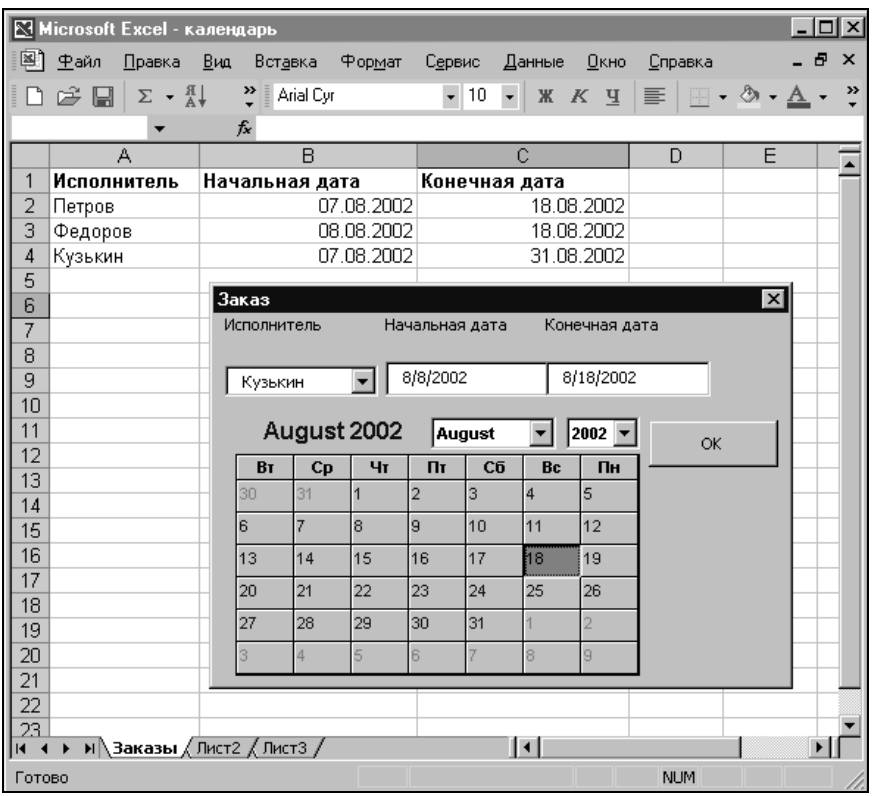


Рис. 7.36. Окно **Заказы**

Листинг 7.50. Заказы

```
Private M As Byte
Private tb(1 To 2) As Object

Private Sub UserForm_Initialize()
   Set tb(1) = txtBegin
   Set tb(2) = txtEnd
   M = 1
   tb(1).SetFocus
   cldMy.Value = Now
   Dim lst As Variant
   lst = Array("Петров", "Кузькин", "Федоров")
   cmbName.List = lst
End Sub

Private Sub txtBegin_Enter()
   M = 1
End Sub

Private Sub txtEnd_Enter()
   M = 2
End Sub

Private Sub cldMy_Click()
   tb(M) = cldMy.Value
End Sub

Private Sub cmdOK_Click()
   Dim n As Integer
   n = Range("A1").CurrentRegion.Rows.Count + 1
   Cells(n, 1).Value = cmbName.Text
   Cells(n, 2).Value = txtBegin.Text
   Cells(n, 3).Value = txtEnd.Text
End Sub
```

Заполнение табличной базы данных

В MS Excel имеется встроенное средство — форма, диалоговое окно которой отображается выбором команды **Данные | Форма**. Форма позволяет заполнять и редактировать записи в таблице базы данных. Большим неудобством этого средства является то, что каждому полю записи в нем соответствует только поле ввода. При заполнении же баз данных значения некоторых полей часто выбираются из конечной группы альтернатив, например пол может быть мужским или женским, список фамилий сотрудников в отделе

продаж ограничен и т. д. Для ввода подобных альтернативных значений удобнее использовать списки, переключатели, флажки, а не ограничиваться полями ввода, как это имеет место в форме. При этом использование списков, переключателей и флажков, во-первых, ускорит процесс заполнения таблицы, а во-вторых, избавит от опечаток, которые неизбежны при ручном вводе данных.

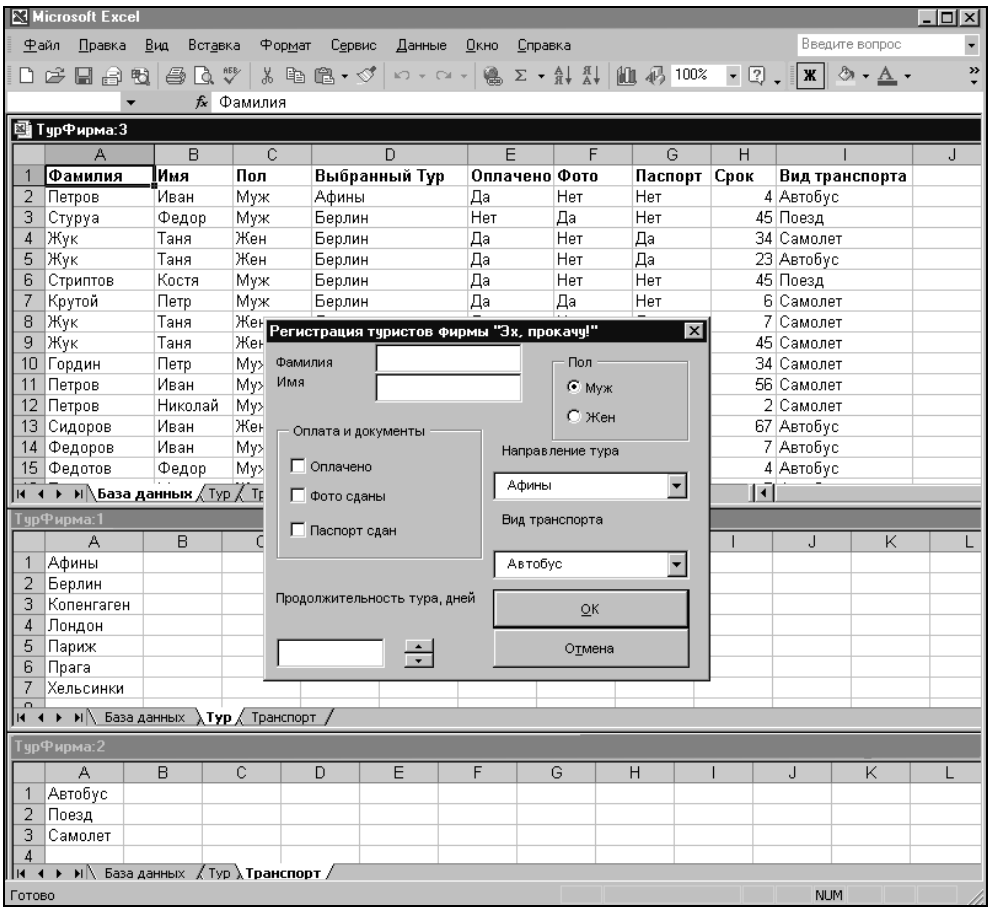


Рис. 7.37. Заполнение табличной базы данных

Создадим приложение, которое лишено указанного недостатка формы. Предположим, что вы — менеджер туристической фирмы "Эх, прокачу!" и заносите информацию о каждом клиенте в базу данных. Для ускорения ввода данных вы решили создать приложение с диалоговым окном. На рис. 7.37 приведена таблица базы данных и диалоговое окно, которое нами будет создано. Во избежание ошибок при вводе данных повторяющая ин-

формация, такая как направление тура и вид транспорта, вводится из списков. Списки же заполняются на основе данных, собранных на рабочих листах **Тур** и **Транспорт**.

Итак, перейдем к конструированию приложения. Создайте форму, а на ней расположите три поля ввода, пять надписей, счетчик, две кнопки, два поля со списком, одну рамку, содержащую два переключателя, а другую — три флажка. При помощи окна **Properties** установите им значения свойств, как показано в табл. 7.29.

Таблица 7.29. *Значения свойств, установленные в окне* ***Properties***

Объект	Свойство	Значение
Форма	Caption	Регистрация туристов фирмы "Эх, прокачу!"
Надпись	Caption	Фамилия
Поле ввода	Name	txtLName
Надпись	Caption	Имя
Поле ввода	Name	txtFName
Рамка	Caption	Оплата и документы
Флажок	Caption	Оплачено
	Name	chkPayment
Флажок	Caption	Фото сданы
	Name	chkPhoto
Флажок	Caption	Паспорт сдан
	Name	chkPassport
Надпись	Caption	Продолжительность тура, дней
Поле ввода	Name	txtDays
Счетчик	Name	spnDays
Рамка	Caption	Пол
Переключатель	Name	optMale
	Caption	Муж
Переключатель	Name	optFemale
	Caption	Жен
Надпись	Caption	Направление тура
Поле со списком	Name	cmbTour
Надпись	Caption	Вид транспорта

Таблица 7.29 *(окончание)*

Объект	Свойство	Значение
Поле со списком	Name	cmbTrans
Кнопка	Name	cmdOK
	Caption	OK
Кнопка	Name	cmdCancel
	Caption	Отмена

В модуле формы наберите следующий код (листинг 7.51), который обеспечивает:

- считывание данных из диалогового окна и ввод записи в первую пустую строку таблицы;
- ввод продолжительности тура либо с клавиатуры, либо из окна счетчика, которые работают синхронно;
- защиту всех данных на рабочих листах от изменений со стороны пользователя.

Листинг 7.51. Заполнение базы данных

```
Private Sub UserForm_Initialize()
   With cmdOK
      .Default = True
      .Accelerator = "O"
      .ControlTipText = "Ввод записи в таблицу"
   End With

   With cmdCancel
      .Cancel = True
      .Accelerator = "т"
      .ControlTipText = "Закрытие окна"
   End With
   optMale.Value = True

   Worksheets("Транспорт").Protect "1", UserInterfaceOnly:=True
   Worksheets("Тур").Protect "1", UserInterfaceOnly:=True
   Worksheets("База данных").Protect Password:="1", _
                                     UserInterfaceOnly:=True
   Dim r As Range
   Set r = Worksheets("Тур").Range("A1").CurrentRegion
```

```
   With cmbTour
      .RowSource = r.Address(RowAbsolute:=False, _
                             ColumnAbsolute:=False, _
                             External:=True)
      .ListIndex = 0
   End With

   Set r = Worksheets("Транспорт").Range("A1").CurrentRegion
   With cmbTrans
      .RowSource = r.Address(RowAbsolute:=False, _
                             ColumnAbsolute:=False, _
                             External:=True)
      .ListIndex = 0
   End With

End Sub

Private Sub cmdOK_Click()
   Dim r As Long
   Dim fname As String
   Dim lname As String
   Dim sex As String
   Dim Tour As String
   Dim HasPaid As String
   Dim HasPhoto As String
   Dim HasPass As String
   Dim Term As String
   Dim Trans As String
   r = WorksheetFunction.CountA(Range("A:A")) + 1

   fname = txtLName.Text
   lname = txtFName.Text
   Term = txtDays.Text
   If optMale.Value Then sex = "Муж"
   If optFemale.Value Then sex = "Жен"
   If chkPayment.Value Then
        HasPaid = "Да"
   Else
        HasPaid = "Нет"
   End If
   If chkPhoto.Value Then
        HasPhoto = "Да"
   Else
        HasPhoto = "Нет"
   End If
```

```
   If chkPassport.Value Then
        HasPass = "Да"
   Else
        HasPass = "Нет"
   End If
    Tour = cmbTour.Text
    Trans = cmbTrans.Text

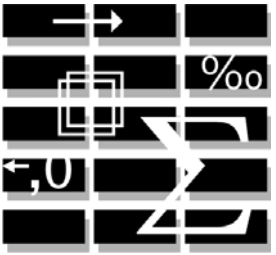
    Cells(r, 1).Value = fname
    Cells(r, 2).Value = lname
    Cells(r, 3).Value = sex
    Cells(r, 4).Value = Tour
    Cells(r, 5).Value = HasPaid
    Cells(r, 6).Value = HasPhoto
    Cells(r, 7).Value = HasPass
    Cells(r, 8).Value = Term
    Cells(r, 9).Value = Trans
End Sub

Private Sub spnDays_Change()
   txtDays.Text = CStr(spnDays.Value)
End Sub

Private Sub txtDays_Change()
   If Not IsNumeric(txtDays.Text) Then Exit Sub
   spnDays.Value = CInt(txtDays.Text)
End Sub

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

ГЛАВА 8



Диаграммы

MS Excel предоставляет в распоряжение разработчика мощное и гибкое средство построения диаграмм — мастер диаграмм. В его арсенале имеется несколько десятков шаблонов плоских и объемных диаграмм, способных удовлетворить любые потребности пользователя для представления данных в наиболее доступном и наглядном графическом виде. Эта глава как раз и посвящена демонстрации возможностей мастера диаграмм, а также объектов `Chart` и `ChartObject`, которые позволяют автоматизировать как процесс построения диаграмм, так и их настройки, в зависимости от бизнес-логики разрабатываемого проекта.

Построение шаблона отчета с диаграммой

В *разд. "Построение шаблона таблицы" главы 1* мы научились строить процедуры, создающие шаблоны отчетов на примере отчета фирмы ООО "Бескрайние просторы". Если бы отчет, дополнительно к таблице, содержал графическое представление данных в виде диаграммы, то это улучшило бы восприятие данных, собранных в отчете, придало бы им большую наглядность и прозрачность. Этот раздел как раз и будет посвящен тому, как научиться создавать процедуры, которые автоматически по таблице данных строят диаграмму (рис. 8.1).

Итак:

1. Выберите команду **Вставка | Диаграмма**.
2. В появившемся окне **Тип диаграммы** мастера диаграмм имеются две вкладки — **Стандартные** со списком стандартных диаграмм (рис. 8.2, *а*) и **Нестандартные** с перечнем дополнительных диаграмм (рис. 8.2, *б*). Диаграммы группируются по типу и виду. На вкладке **Стандартные** в списке

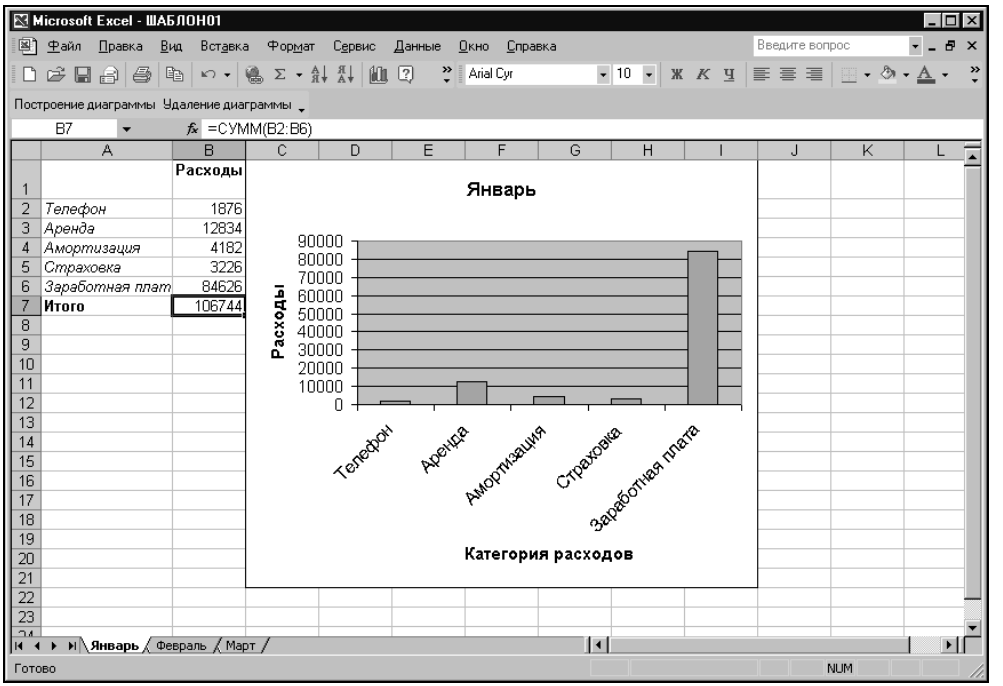
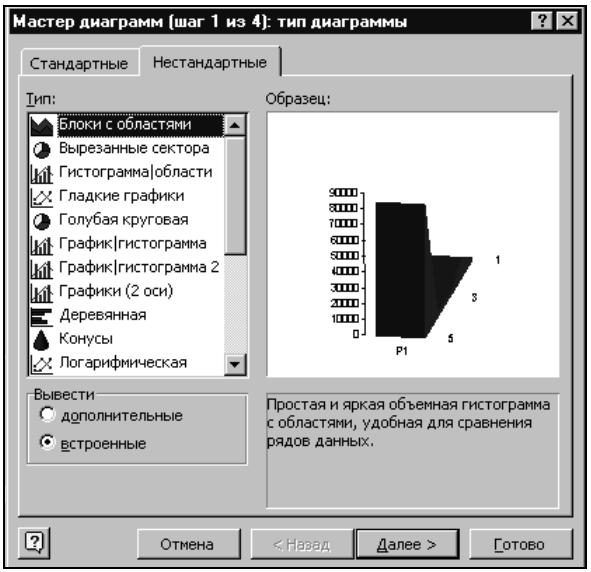


Рис. 8.1. Шаблон отчета с диаграммой



а

Рис. 8.2. Окно **Тип диаграммы** *(см. продолжение)*



б

Рис. 8.2. *Окончание*

Тип выберите **Гистограмма**, а в списке **Вид** выберите обычную гистограмму. Нажмите кнопку **Далее**.

3. В появившемся диалоговом окне **Источник данных диаграммы** мастера диаграмм на вкладке **Диапазон данных** выберите в группе **Ряды в** переключатель **столбцах**, т. к. данные располагаются в столбцах, а в поле ввода **Диапазон** приведите ссылку на диапазон данных, значения которого откладываются по оси ординат (рис. 8.3). В данном случае, на диапазон `=Январь!$B$2:$B$6`.

4. На вкладке **Ряд** окна **Исходные данные** (рис. 8.4) в поле ввода **Подписи оси X** приведите ссылку на диапазон `=Январь!$A$2:$A$6`, значения из которого откладываются по оси абсцисс. Если в этом поле ввода не приводить ссылку ни на какой диапазон, то данные, откладываемые по оси ординат, нумеруются в естественном порядке начиная с 1, а по оси абсцисс будут откладываться эти порядковые номера. В списке **Ряд** приводятся ряды данных, откладываемых по оси ординат (в нашем случае имеется только один ряд данных). Эти ряды автоматически определяются на основе ссылки, указанной в поле ввода **Диапазон** предыдущего шага алгоритма. В поле **Значения** автоматически выводится ссылка на диапазон, соответствующий выбранному ряду из списка **Ряд**. В поле ввода **Имя** приводится ссылка на ячейку, в которой содержится заголовок соответствующего ряда. Этот заголовок в последующем используется мастером диаграмм для создания легенды. Легенда в диаграмме требуется для того, чтобы различать несколько рядов данных, откладываемых по

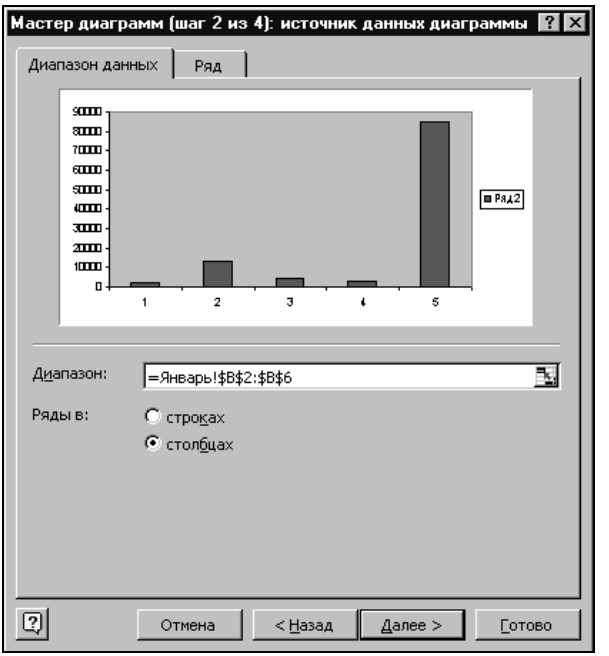


Рис. 8.3. Вкладка **Диапазон данных** окна **Источник данных диаграммы**

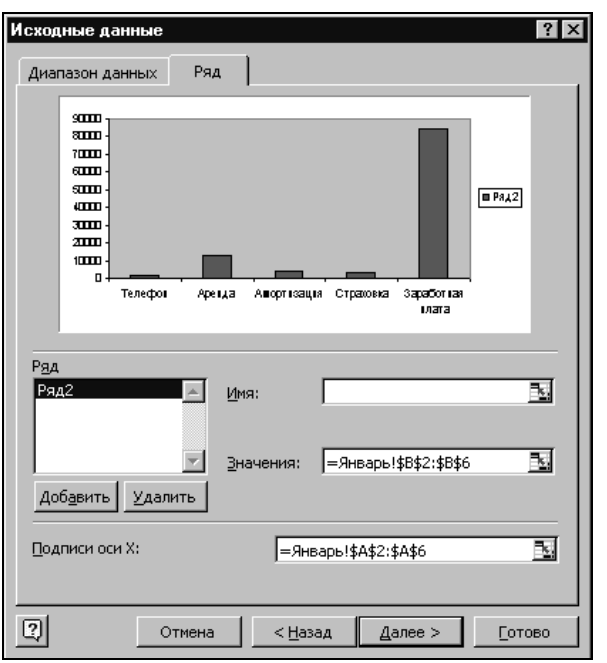


Рис. 8.4. Вкладка **Ряд** диалогового окна **Исходные данные**

оси ординат. В нашем случае имеется только один ряд данных, поэтому легенда нам не потребуется. Следовательно, в поле ввода **Имя** вводить ничего не надо. Нажмите кнопку **Далее**.

5. В появившемся окне **Параметры диаграммы** мастера диаграмм на вкладке **Заголовки** (рис. 8.5, *а*) в поле **Название диаграммы** введите `Январь`, в поле **Ось X (категорий)** введите `Категория расходов`, в поле **Ось Y (значений)** введите `Расходы`. На вкладке **Легенда** (рис. 8.5, *б*) снимите флажок **Добавить легенду**. Нажмите кнопку **Далее**.

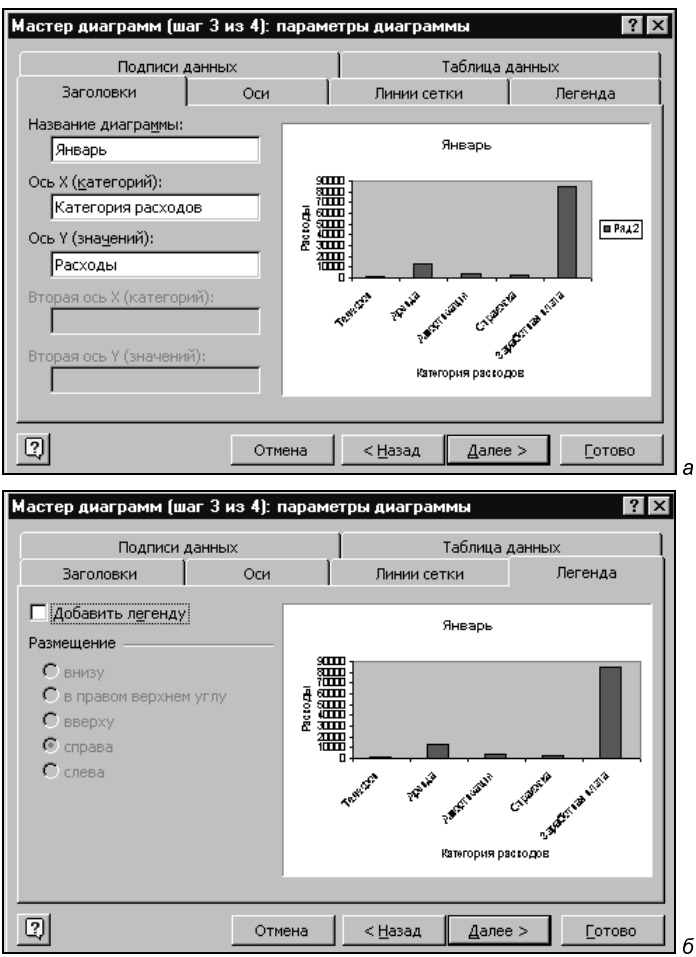


Рис. 8.5. Вкладки **Заголовки** и **Легенда** окна **Параметры диаграммы**

6. В появившемся диалоговом окне **Размещение диаграммы** мастера диаграмм (рис. 8.6) в группе **Поместить диаграмму на листе** выберите переключатель **имеющемся**. Диаграмма будет внедрена в рабочий лист, имя

которого указывается в соответствующем списке. Если выбрать переключатель **отдельном**, то диаграмма будет помещена на листе диаграмм. Нажмите кнопку **Готово**.

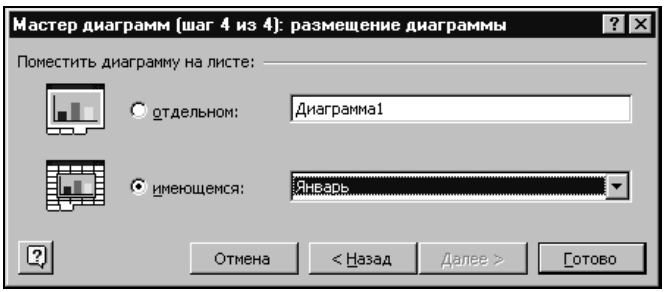


Рис. 8.6. Диалоговое окно **Размещение диаграммы**

7. Диаграмма успешно внедрена в рабочий лист. Используя маркеры перемещения и изменения размера, ей можно придать требуемые размеры и разместить в нужном месте.

Семейства *ChartObjects* и *Charts* и объекты *ChartObject* и *Chart*

MS Excel позволяет создавать высококачественные диаграммы. Существенную помощь при построении диаграмм оказывают встроенные форматы диаграмм, которых несколько десятков. Можно найти встроенный формат диаграммы на любой случай жизни. Как вся диаграмма, так и ее отдельные элементы легко удаляются, изменяются и добавляются. Добавляя в диаграмму пояснительный текст, стрелки, заголовки, легенду, изменяя заливку, узоры, типы линий и обрамление, можно добиться того, чтобы диаграмма представляла данные в наиболее наглядном виде. В MS Excel имеются два типа диаграмм:

- внедренные в рабочий лист;
- созданные на специальном листе диаграмм.

Семейство `Sheets` листов рабочей книги включает в себя два семейства листов: `Worksheets` (рабочих листов) и `Charts` (листы диаграмм). Объектами семейства `Charts` являются диаграммы, созданные на листах диаграмм. Это семейство не включает в себя диаграммы, непосредственно внедренные в рабочие листы. Такие диаграммы принадлежат семейству `ChartObjects`. Таким образом, объект `ChartObject` встроен в объект `Worksheet`, а `Chart` — в `Workbook`. У объектов `Workbook` и `Application` имеется общее свойство `ActiveChart`, которое возвращает активную диаграмму рабочей книги, независимо от того, какому семейству она принадлежит. У объекта `Chart` имеет-

ся ряд дочерних по отношению к нему объектов, которые перечислены в табл. 8.1.

Таблица 8.1. *Подчиненные объекту* `Chart` *объекты*

Объект	Описание
`ChartArea`	Область, в которой нарисована диаграмма
`PlotArea`	Область диаграммы, где нарисована диаграмма
`Floor`	Горизонтальная плоскость трехмерной диаграммы
`Walls`	Вертикальные плоскости трехмерной диаграммы
`Corners`	Углы трехмерной диаграммы
`PageSetup`	Параметры страницы
`ChartTitle`	Заголовок диаграммы
`SeriesCollection`	Диапазон данных, откладываемых по оси ординат
`Tredlines`	Линия тренда
`Axis`	Оси координат
`AxisTitle`	Заголовки осей
`DisplayUnitLabel`	Единица масштабирования осей
`Gridlines`	Координатная сетка
`TickLabels`	Значки, откладываемые по осям
`DataTable`	Таблица данных диаграммы
`Legend`	Легенда

Добавление нового элемента в семейства *ChartObjects* и *Charts*

Как и любые семейства, семейства `ChartObjects` и `Charts` имеют методы `Add` (создает новый элемент семейства), `Delete` (удаляет элемент семейства) и свойство `Count` (возвращает число элементов семейства).

Метод `Add` семейства `ChartObjects`:

`Add(`*`Left`*`, `*`Top`*`, `*`Width`*`, `*`Height`*`)`

- *`Left`*, *`Top`* — задают координаты на рабочем листе левого верхнего угла диаграммы;
- *`Width`*, *`Height`* — устанавливают ширину и высоту диаграммы.

Все параметры метода необязательные.

Метод `Add` семейства `Charts`:

`Add(`*`Before`*`, `*`After`*`, `*`Count`*`)`

- *`Before`* — задает, перед каким листом добавляется диаграмма;
- *`After`* — указывает, после какого листа добавляется диаграмма;
- *`Count`* — определяет число добавляемых диаграмм.

Все параметры метода необязательные.

Свойства объекта *Chart*

Объект `Chart` имеет более 50 свойств, определяющих внешний вид диаграммы (подробное описание этих свойств можно найти в справочной системе MS Excel). В табл. 8.2 перечислены основные свойства объекта `Chart`.

Таблица 8.2. *Основные свойства объекта* `Chart`

Свойство	Описание
`Area3DGroup`	Возвращает объект `ChartGroup`, инкапсулирующий информацию об области, в которой выводится трехмерная диаграмма
`AutoScaling`	Устанавливает автоматическое масштабирование трехмерных диаграмм
`Bar3DGroup`	Возвращает объект `ChartGroup`, инкапсулирующий информацию о трехмерной диаграмме
`ChartArea`	Возвращает объект `ChartArea`
`ChartTitle`	Возвращает объект `ChartTitle`
`ChartType`	Задает тип диаграммы. Допустимые значения приведены в табл. 8.3
`Column3DGroup`	Возвращает объект `ChartGroup`, инкапсулирующий информацию о столбцах трехмерной диаграммы
`Corners`	Возвращает объект `Corners`
`DataTable`	Возвращает объект `DataTable`
`DepthPercent`	Устанавливает коэффициент глубины для трехмерной диаграммы
`DisplayBlanksAs`	Задает, как пустые ячейки интерпретируются при построении диаграммы. Допустимые значения: `xlNotPlotted`, `xlInterpolated` и `xlZero`

Таблица 8.2 (окончание)

Свойство	Описание
Elevation	Задает угол, под которым смотрят на трехмерную диаграмму
Floor	Возвращает объект Floor
GapDepth	Задает промежуток между рядами для трехмерной диаграммы
HasAxis	Устанавливает, имеются ли у диаграммы оси
HasDataTable	Устанавливает, имеется ли таблица данных
HasLegend	Проверяет, имеется ли легенда
HasTitle	Проверяет, имеется ли заголовок
HeightPercent	Задает высоту диаграммы как процент по отношению к ее ширине
Hyperlinks	Возвращает семейство Hyperlinks
Index	Возвращает значение индекса в семейства Charts
Legend	Возвращает объект Legend
PageSetup	Возвращает объект PageSetup
Perspective	Устанавливает перспективу для трехмерной диаграммы
PlotArea	Возвращает объект PlotArea
PlotBy	Определяет, как данные располагаются в диапазоне. Допустимые значения: xlColumns и xlRows
PlotVisibleOnly	Задает, надо ли учитывать невидимые ячейки
ProtectContents, ProtectData, ProtectDrawingObjects, ProtectFormatting, ProtectionSelection, ProtectGoalSeek	Логические свойства, которые определяют, установлена ли защита на соответствующий элемент диаграммы
Rotation	Возвращает угол поворота 3-мерной диаграммы вокруг оси Z
Visible	Управляет видимостью диаграммы
Walls	Возвращает объект Walls

Таблица 8.3. *Допустимые значения свойства* `ChartType`

Тип диаграммы	Константы
Гистограмма	`xlColumnClustered`, `xl3DcolumnClustered`, `xlColumnStacked`, `xl3DcolumnStacked`, `xlColumnStacked100`, `xl3DColumnStacked100`, `xl3Dcolumn`
Линейчатая	`xlBarClustered`, `xl3DbarClustered`, `xlBarStacked`, `xl3DbarStacked`, `xlBarStacked100`, `xl3DbarStacked100`
График	`xlLine`, `xlLineMarkers`, `xlLineStacked`, `xlLineMarkersStacked`, `xlLineStacked100`, `xlLineMarkersStacked100`, `xl3Dline`
Круговая	`xlPie`, `xlPieExploded`, `xl3Dpie`, `xl3DpieExploded`, `xlPieOfPie`, `xlBarOfPie`
Точечная	`xlXYScatter`, `xlXYScatterSmooth`, `xlXYScatterSmoothNoMarkers`, `xlXYScatterLines`, `xlXYScatterLinesNoMarkers`
С областями	`xlArea`, `xl3Darea`, `xlAreaStacked`, `xl3DareaStacked`, `xlAreaStacked100`, `xl3DAreaStacked100`
Кольцевая	`xlDoughnut`, `xlDoughnutExploded`
Лепестковая	`xlRadar`, `xlRadarMarkers`, `xlRadarFilled`
Поверхность	`xlSurface`, `xlSurfaceTopView`, `xlSurfaceWireframe`, `xlSurfaceTopViewWireframe`
Пузырьковая	`xlBubble`, `xlBubble3Deffect`
Биржевая	`xlStockHLC`, `xlStockVHLC`, `xlStockOHLC`, `xlStockVOHLC`
Цилиндрическая	`xlCylinderColClustered`, `xlCylinderBarClustered`, `xlCylinderColStacked`, `xlCylinderBarStacked`, `xlCylinderColStacked100`, `xlCylinderBarStacked100`, `xlCylinderCol`
Коническая	`xlConeColClustered`, `xlConeBarClustered`, `xlConeColStacked`, `xlConeBarStacked`, `xlConeColStacked100`, `xlConeBarStacked100`, `xlConeCol`
Пирамидальная	`xlPyramidColClustered`, `xlPyramidBarClustered`, `xlPyramidColStacked`, `xlPyramidBarStacked`, `xlPyramidColStacked100`, `xlPyramidBarStacked100`, `xlPyramidCol`

Методы объекта *Chart*

Объект `Chart`, как и любой объект, имеет не только свойства, но и методы, позволяющие управлять его внешним видом. В табл. 8.4 перечислены методы объекта `Chart`.

Таблица 8.4. *Методы объекта* `Chart`

Метод	Описание
`Activate`	Активизация диаграммы
`ApplyDataLabels`	Применение специфицированных меток
`AutoFormat`	Автоформат
`Axes`	Возвращает семейство `Axes`, используемое для установки различных свойств осей
`ChartObjects`	Возвращает семейство `ChartObjects`
`ChartWizard`	Построение диаграммы
`CheckSpelling`	Проверка орфографии
`Copy`	Копирует диаграмму в указанное местоположение
`CopyPicture`	Копирует диаграмму в буфер обмена как рисунок
`Delete`	Удаляет диаграмму
`Deselect`	Снимает выделение с диаграммы
`Export`	Экспортирует диаграмму в файл графического формата
`GetChartElement`	Возвращает информацию об элементе диаграммы, расположенном в указанной точке
`Location`	Задает местоположение диаграммы
`Move`	Перемещает диаграмму
`Paste`	Вставляет данные диаграммы из буфера обмена
`PrintOut`	Выводит диаграмму на печать
`SendToBack`	Отображает диаграмму на заднем плане
`Protect`	Устанавливает защиту
`Refresh`	Обновляет диаграмму
`SaveAs`	Сохраняет измененную диаграмму в новом файле
`Select`	Выбирает диаграмму
`SeriesCollection`	Возвращает ряды данных
`SetBackgroundPicture`	Задает фоновый рисунок

Таблица 8.4 *(окончание)*

Метод	Описание
SetSourceData	Задает диапазон, на основе которого строится диаграмма
Unprotect	Снимает защиту с диаграммы

События объекта *Chart*

У объекта Chart имеется целый ряд событий (табл. 8.5), позволяющих отслеживать различные действия пользователя.

Таблица 8.5. *События объекта* Chart

Событие	Описание
Activate	Происходит при активизации диаграммы
BeforeDoubleClick	Происходит перед двойным щелчком
BeforeRightClick	Происходит перед щелчком правой кнопкой
Calculate	Происходит при изменении данных
Deactivate	Происходит при деактивизации диаграммы
DragOver	Происходит при буксировке диапазона над диаграммой
DragPlot	Происходит при буксировке и вставке диапазона в диаграмму
MouseDown, MouseUp	Происходят, когда пользователь нажимает и отпускает любую кнопку мыши
MouseMove	Происходит, когда пользователь передвигает указатель мыши над диаграммой
Resize	Происходит при изменении размеров диаграммы
Select	Происходит при выборе элемента диаграммы
SeriesChange	Происходит при изменении указателя на ряд данных

Метод *ChartWizard*

Среди методов объекта Chart особое место занимает метод ChartWizard. Он позволяет модифицировать диаграмму без установки ее свойств по раздельности, т. е. он предоставляет возможность управлять диаграммой в целом.

```
ChartWizard(Source, Gallery, Format, PlotBy, CategoryLabels,
↳SeriesLabels, HasLegend, Title, CategoryTitle, ValueTitle, ExtraTitle)
```

- *Source* — диапазон с данными, на основе которых строится диаграмма;
- *Gallery* — тип диаграммы. Допустимы значения: `xlArea`, `xlBar`, `xlColumn`, `xlLine`, `xlPie`, `xlRadar`, `xlXYScatter`, `xlCombination`, `xl3DArea`, `xl3DBar`, `xl3DColumn`, `xl3DLine`, `xl3Dpie`, `xl3Dsurface`, `xlDoughnut` или `xlDefaultAutoFormat`;
- *Format* — формат для данного типа диаграммы. Каждый тип имеет до десяти форматов;
- *PlotBy* — задает, как располагаются данные в диапазоне, на основе которого строится диаграмма. Допустимые значения: `xlRows` (по строкам) и `xlColumns` (по столбцам);
- *CategoryLabels* — число строк (столбцов) с метками категорий (т. е. данных, отводимых под ось абсцисс);
- *SeriesLabels* — число строк (столбцов) с метками рядов (т. е. данных, отводимых под заголовки);
- *HasLegend* — параметр, принимающий логические значения, устанавливает, надо ли отображать легенду;
- *Title* — заголовок диаграммы;
- *CategoryTitle* — название оси абсцисс;
- *ValueTitle* — название оси Z;
- *ExtraTitle* — название оси ординат.

Построение диаграммы

О том, как построить диаграмму вручную, было рассказано в *разд. "Построение шаблона отчета с диаграммой"*. Теперь пора попытаться построить ее в коде. В данном разделе будет сконструирован следующий проект. При открытии книги создается пользовательская панель инструментов с двумя кнопками **Построение диаграммы** и **Удаление диаграммы**. Нажатием кнопки **Построение диаграммы** производится построение диаграммы на активном рабочем листе, причем ее заголовок будет совпадать с именем активного листа (рис. 8.7).

Построение диаграммы производится по следующим этапам. Сначала, во избежание наложения диаграмм, удаляются все диаграммы, расположенные на активном листе. Затем методом `Add` создается новая диаграмма. Свойством `ChartType` задается тип диаграммы. Методом `SetSourceData` дается ссылка на диапазон, значения из которого откладываются по оси ординат. В данном случае это диапазон **B2:B6** активного листа. Метод `SeriesCollection` задает ссылку на диапазон, значения из которого отклады-

ваются по оси абсцисс. В нашем случае это диапазон **A2:A6** активного листа. Затем методом `Location` указывается местоположение диаграммы. В данном случае она будет внедрена в рабочий лист со специфицированным именем, который по сценарию проекта и является активным листом. После этого свойством `ChartTitle` и методом `Axes` задаются заголовок (который совпадает с именем рабочего листа) и названия осей. В заключение свойством `HasLegend` удаляется легенда и свойствами `Top`, `Left`, `Width` и `Height` диаграмма размещается в специфицированном месте рабочего листа. Таким образом, этот проект позволяет строить диаграмму на любом рабочем листе.

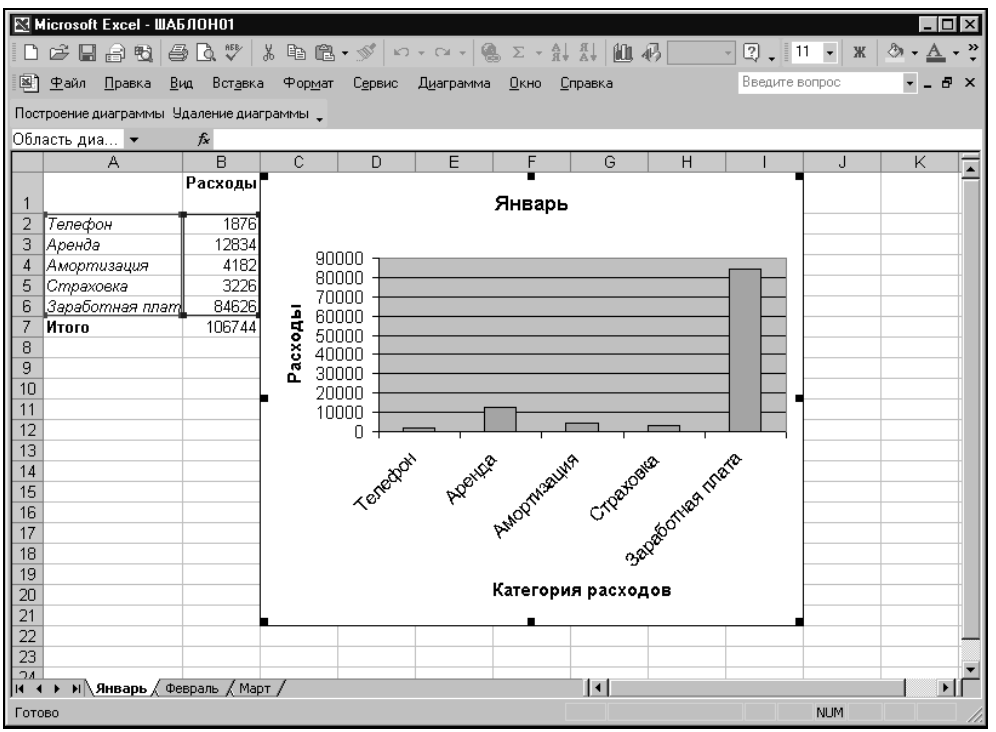


Рис. 8.7. Построение диаграммы

Итак, осталось только набрать в стандартном модуле код, реализующий построение и удаление диаграммы (листинг 8.1), а в модуле `ЭтаКнига` — код, создающий интерфейс проекта (листинг 8.2).

Листинг 8.1. Построение диаграммы. Стандартный модуль

```
Sub ChartBuilder()
    Dim rx As Range
    Dim ry As Range
```

```
   Dim nameX As String
   Dim nameY As String
   Dim title As String
   Dim nameSh As String
   nameX = "Категория расходов"
   nameY = "Расходы"
   nameSh = ActiveSheet.Name
   title = nameSh
   Set ry = Sheets(nameSh).Range("B2:B6")
   Set rx = Sheets(nameSh).Range("A2:A6")
   ChartKiller
   Charts.Add
   ActiveChart.ChartType = xlColumnClustered
   ActiveChart.SetSourceData Source:=ry, PlotBy:=xlColumns
   ActiveChart.SeriesCollection(1).XValues = _
       "=" & rx.Address(ReferenceStyle:=xlR1C1, external:=True)
   ActiveChart.Location Where:=xlLocationAsObject, Name:=nameSh
   With ActiveChart
      .HasTitle = True
      .ChartTitle.Characters.Text = title
      .Axes(xlCategory, xlPrimary).HasTitle = True
      .Axes(xlCategory, xlPrimary).AxisTitle.Characters.Text = nameX
      .Axes(xlValue, xlPrimary).HasTitle = True
      .Axes(xlValue, xlPrimary).AxisTitle.Characters.Text = nameY
   End With
   ActiveChart.HasLegend = False
   With Worksheets(nameSh).ChartObjects(1)
      .Top = Range("C1").Top
      .Left = Range("C1").Left
      .Width = Range("C1:I21").Width
      .Height = Range("C1:I21").Height
   End With
End Sub

Sub ChartKiller()
   ActiveSheet.ChartObjects.Delete
End Sub
```

Листинг 8.2. Построение диаграммы. Модуль ЭтаКнига

```
Private Sub Workbook_Open()
   Dim cbMenuBar As CommandBar
```

```
    Dim btnPlot As CommandBarButton
    Dim btnKill As CommandBarButton
    Set cbMenuBar = Application.CommandBars.Add(Name:="МояДиаграмма", _
                                                Position:=msoBarTop, _
                                                MenuBar:=False, _
                                                Temporary:=True)
    With cbMenuBar
        .Visible = True
        .Protection = msoBarNoMove + msoBarNoChangeVisible _
                    + msoBarNoCustomize
        With .Controls
            Set btnPlot = .Add(Type:=msoControlButton)
            Set btnKill = .Add(Type:=msoControlButton)
        End With
    End With

    With btnPlot
        .Caption = "Построение диаграммы"
        .Style = msoButtonCaption
        .OnAction = "ChartBuilder"
    End With
    With btnKill
        .Caption = "Удаление диаграммы"
        .Style = msoButtonCaption
        .OnAction = "ChartKiller"
    End With
End Sub

Private Sub Workbook_BeforeClose(Cancel As Boolean)
    Application.CommandBars("МояДиаграмма").Delete
End Sub
```

Изменение диапазона, по которому строится диаграмма

В *разд. "Построение шаблона таблицы" главы 1* была построена итоговая таблица квартальных расходов фирмы ООО "Бескрайние просторы". Для большей наглядности мы добавим в это приложение диаграмму, которая показывает динамику расходов по каждой из шести категорий (рис. 8.8). Список категорий будет выводиться в элементе управления **Список**. Выбор категории из списка будет автоматически приводить к изменению диапазона, по которому строится диаграмма, и соответствующей перестройке диаграммы.

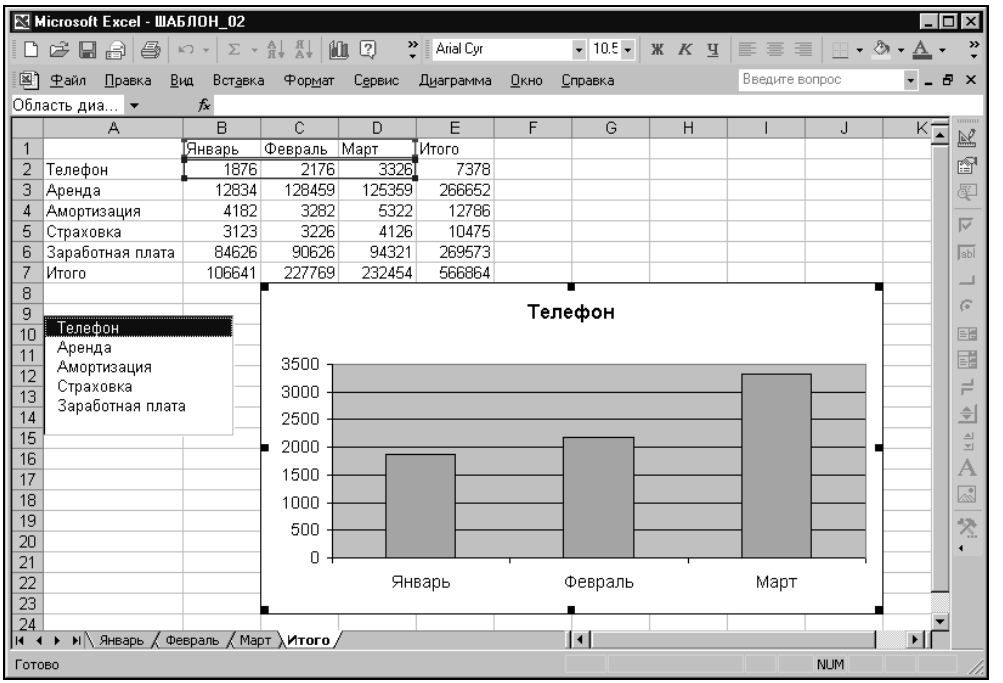


Рис. 8.8. Изменение диапазона, по которому строится диаграмма

На рабочем листе создайте список и, используя окно **Properties,** установите значение его свойства `Name` равным `lstCategory`. В модуле `ЭтаКнига` наберите приводимый ниже код (листинг 8.3, *а*). Процедура `Workbook_Open` выполняется при открытии рабочей книги. Она удаляет все диаграммы с рабочего листа **Итого**, строит гистограмму по расходам за телефон и располагает ее так, чтобы она занимала диапазон **C8:J23**. Это гарантирует неналожение диаграммы на таблицу с данными. В процедуре также заполняется список на основе заголовков записей таблицы данных и происходит выбор первого элемента из этого списка.

Листинг 8.3, *а*. Выбор категории расходов. Модуль `ЭтаКнига`

```
Private Sub Workbook_Open()
   DeleteCharts
   ChartBuilder
   FilllstCategory
End Sub

Private Sub DeleteCharts()
   With Worksheets("Итого")
```

```
      If .ChartObjects.Count > 0 Then
          .ChartObjects.Delete
      End If
   End With
End Sub

Private Sub ChartBuilder()
   Charts.Add
   With ActiveChart
      .ChartType = xlColumnClustered
      .SetSourceData _
          Source:=Sheets("Итого").Range("B2:D2"), PlotBy:=xlRows
      .SeriesCollection(1).XValues = "= Итого!R1C2:R1C4"
      .Location Where:=xlLocationAsObject, Name:="Итого"
   End With
   With ActiveChart
      .HasTitle = True
       .ChartTitle.Characters.Text = _
          Worksheets("Итого").Range("A2").Value
       .Axes(xlCategory, xlPrimary).HasTitle = False
       .Axes(xlValue, xlPrimary).HasTitle = False
   End With
   ActiveChart.HasLegend = False
   With Worksheets("Итого").ChartObjects(1)
      .Top = Range("C8").Top
       .Left = Range("C8").Left
       .Width = Range("C8:J23").Width
       .Height = Range("C8:J23").Height
   End With
End Sub

Private Sub FilllstCategory()
   With Worksheets("Итого").lstCategory
      .ListFillRange = "Итого!A2:A6"
      .ListIndex = 0
   End With
End Sub
```

Для завершения создания приложения в модуле рабочего листа **Итого** наберите следующую процедуру обработки события `Click` списка, которая при выборе категории расходов из списка перестраивает диаграмму (листинг 8.3, *б*).

Листинг 8.3, *б*. Выбор категории расходов. Модуль рабочего листа Итого

```
Private Sub lstCategory_Click()
   Dim r As Integer
   ActiveSheet.ChartObjects(1).Activate
   r = lstCategory.ListIndex + 1
   With ActiveChart
     .SetSourceData _
         Source:= _
           Sheets("Итого").Range("B2:D6").Rows(r), PlotBy:=xlRows
      .SeriesCollection(1).XValues _
          = Sheets("Итого").Range("B1:D1")
   End With
   With ActiveChart
      .HasTitle = True
      .ChartTitle.Characters.Text = lstCategory.Text
   End With
End Sub
```

Изменение типа диаграммы

Продолжим модифицирование приложения из предыдущего раздела — построение итоговой таблицы квартальных расходов фирмы ООО "Бескрайние просторы". Усовершенствуем его до уровня, когда пользователь сможет управлять как типом диаграммы, так и отображением легенды (рис. 8.9).

Итак, перейдем к модификации приложения. На рабочем листе **Итого** создайте еще один список. При помощи окна **Properties** установите значение его свойства `Name` равным `lstType`, в модуле рабочего листа **Итого** наберите код процедуры `FilllstType` (листинг 8.3, *в*), а в процедуру обработки события `Open` рабочей книги из предыдущего раздела добавьте вызов процедуры `FilllstType`.

Листинг 8.3, *в*. Управление типом и легендой диаграммы. Модуль `ЭтаКнига`

```
Private Sub Workbook_Open()
   DeleteCharts
   ChartBuilder
   FilllstCategory
   FilllstType
End Sub

Private Sub FilllstType()
   Dim tb(2, 1) As Variant
```

```
    tb(0, 0) = "Гистограмма"
    tb(0, 1) = xlColumnClustered
    tb(1, 0) = "График" : tb(1, 1) = xlLine
    tb(2, 0) = "Круговая" : tb(2, 1) = xlPie
    With Worksheets("Итого").lstType
        .ColumnCount = 2
        .TextColumn = 2
        .ColumnWidths = "70;0"
        .List = tb
        .ListIndex = 0
    End With
End Sub
```

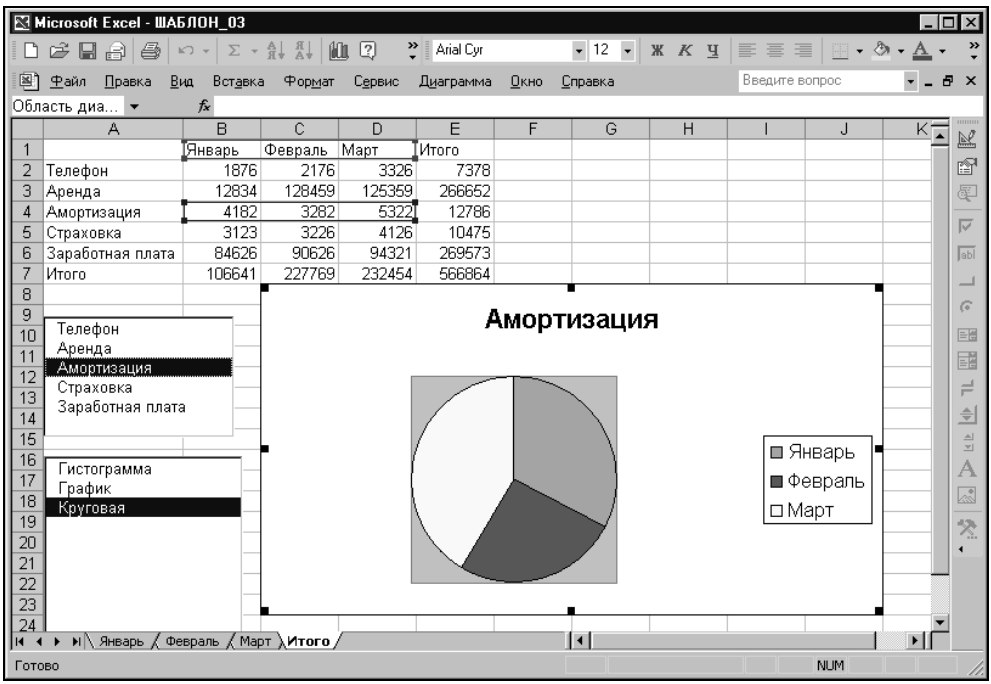


Рис. 8.9. Управление типом и легендой диаграммы

Для завершения создания приложения в модуле рабочего листа **Итого** наберите следующую процедуру обработки события `Click` списка, которая при выборе типа диаграммы перестраивает ее (листинг 8.3, *г*).

Листинг 8.3, *г*. Выбор категории расходов. Модуль рабочего листа Итого

```
Private Sub lstType_Click()
    ActiveSheet.ChartObjects(1).Activate
```

```
   ActiveChart.ChartType = lstType.Text
   If lstType.Text = xlPie Then
      ActiveChart.HasLegend = True
   Else
      ActiveChart.HasLegend = False
   End If
End Sub
```

Примечание

Обратите внимание на один трюк, использованный в данном приложении. У диаграмм есть название типов, а также константы, задающие типы диаграмм. В списке нам надо отобразить только названия типов диаграмм, а результатом выбора элемента из списка должна быть константа, задающая этот тип. В коде данная ситуация разрешается следующим образом. Создается двухстолбцовый список. Первым столбец состоит из названий типов, а второй из констант, задающих эти типы. Второй столбец нам не нужен, поэтому ему устанавливается нулевая ширина, что приводит к тому, что элементы второго столбца не отображаются в списке. Но т. к. значение свойства `TextColumn` списка установлено равным 2, то при выборе элемента из списка в качестве значения свойства `Text` возвращается элемент второго, а не первого столбца. Чего, собственно говоря, и стремились добиться.

Примечание

В зависимости от типа диаграммы легенда может быть необходима, а может быть излишеством, только затрудняющим восприятие данных, приведенных в диаграмме. Например, для гистограммы легенда в построенном приложении совсем не нужна, но для круговой диаграммы без легенды не обойтись. Поэтому в процедуре `lstType_Click` в зависимости от выбранного типа диаграммы легенда либо отображается, либо не отображается.

Автоматическое определение диапазона, по которому строится диаграмма

Сделаем еще один шаг в направлении улучшения создаваемого приложения по построению итоговой таблицы расходов фирмы ООО "Бескрайние просторы". В предыдущих разделах это приложение было привязано к количеству месяцев (трем месяцам), за которые подводятся итоги. В данном разделе мы снимем это ограничение, т. е. в итоговой таблице может быть произвольное число месяцев (они могут добавляться в таблицу или, наоборот, исключаться из нее). После изменения количества месяцев пользователю достаточно щелкнуть на списке с категориями расходов, и диаграммы будут автоматически перестроены. Для реализации этого проекта надо видоизменить процедуру обработки события `Click` списка категорий расходов следующим образом (листинг 8.4).

Листинг 8.4. Управление типом и легендой диаграммы. Модуль рабочего листа Итого

```
Private Sub lstCategory_Click()
   Dim r As Integer
   ActiveSheet.ChartObjects(1).Activate
   r = lstCategory.ListIndex + 1
   Dim rgn As Range
   Dim rgnTitle As Range
   Set rgn = Range("A1").CurrentRegion

   Set rgnTitle = rgn.Rows(1)
   Set rgnTitle = rgnTitle.Offset(0, 1)
   Set rgnTitle = rgnTitle.Resize( _
              ColumnSize:=rgnTitle.Columns.Count - 2)

   Set rgn = rgn.Offset(1, 1)
   Set rgn = rgn.Resize(rgn.Rows.Count - 2, _
                        rgn.Columns.Count - 2)

   With ActiveChart
      .SetSourceData Source:=rgn.Rows(r), _
                     PlotBy:=xlRows
       .SeriesCollection(1).XValues = rgnTitle
   End With
   With ActiveChart
      .HasTitle = True
      .ChartTitle.Characters.Text = lstCategory.Text
   End With
End Sub
```

Линия тренда

С помощью диаграммы можно легко анализировать тренды и делать прогнозы. Вручную на листе линия тренда к диаграмме добавляется командой **Диаграмма | Добавить линию тренда** (предварительно, конечно, надо выбрать саму диаграмму, и тогда меню **Данные** заменится на меню **Диаграмма**).

Все линии тренда, соответствующие конкретному ряду данных, образуют семейство `Trendlines`, элементами которого являются объекты `Trendline` (линия тренда). Семейство `Trendlines` имеет только два метода: `Add`, добавляющий новый элемент в семейство, и `Item`, возвращающий конкретный элемент семейства.

Приведем описание метода `Add` семейства `Trendlines`.

```
Add(Type, Order, Period, Forward, Backward, Intercept, DisplayEquation,
↳DisplayRSquared, Name)
```

- *Type* — устанавливает тип линии тренда. Допустимые значения: `xlLinear` (линейная), `xlLogarithmic` (логарифмическая), `xlExponential` (экспоненциальная), `xlPolynomial` (полиномиальная), `xlMovingAvg` (скользящее среднее) и `xlPower` (степенная);
- *Order* — устанавливает порядок линии тренда. Допустимые значения: целые числа из интервала от 2 до 6. Используется, если аргумент `Type` принимает значение `xlPolynomial`;
- *Period* — период тренда. Допустимые значения: целое из диапазона от 1 до числа точек, по которым строится тренд. Используется, если аргумент `Type` принимает значение `xlMovingAvg`;
- *Forward* — число точек вперед (в будущее) для предсказания значений в соответствии с трендом;
- *Backward* — число точек назад (в прошлое) для предсказания значений в соответствии с трендом;
- *Intercept* — пересечение с осью ординат;
- *DisplayEquation* — параметр, принимающий логические значения, показывающие, надо ли отображать на диаграмме уравнение тренда;
- *DisplayRSquared* — параметр, принимающий логическое значение, показывающее, надо ли отображать квадрат коэффициента корреляции;
- *Name* — строка, задающая имя линии тренда.

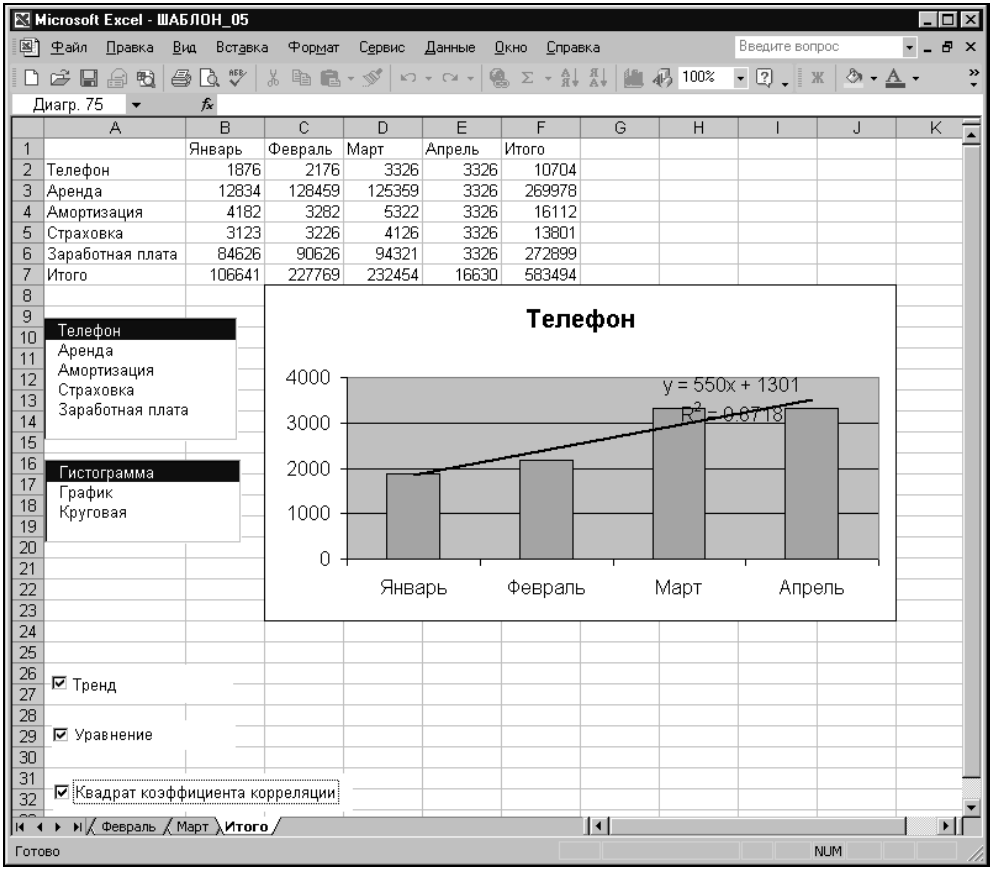
Примечание

Объект `Trendline` имеет такие же свойства, как и только что перечисленные параметры метода `Add`.

Проект с линией тренда

Сделаем еще один шаг в направлении улучшения создаваемого приложения по построению итоговой таблицы расходов фирмы ООО "Бескрайние просторы". Добавим в наш проект дополнительное средство, образованное группой из трех флажков, позволяющее управлять отображением линии тренда в диаграмме (рис. 8.10).

На рабочий лист **Итого** добавьте три флажка и при помощи окна **Properties** установите им значения свойств, как показано в табл. 8.6.



	A	B	C	D	E	F
1		Январь	Февраль	Март	Апрель	Итого
2	Телефон	1876	2176	3326	3326	10704
3	Аренда	12834	128459	125359	3326	269978
4	Амортизация	4182	3282	5322	3326	16112
5	Страховка	3123	3226	4126	3326	13801
6	Заработная плата	84626	90626	94321	3326	272899
7	Итого	106641	227769	232454	16630	583494

Рис. 8.10. Проект с линией тренда

Таблица 8.6. *Значения свойств, установленные в окне* ***Properties***

Элемент управления	Свойство	Значение
Флажок	Name	chkTrend
	Caption	Тренд
Флажок	Name	chkEq
	Caption	Уравнение
Флажок	Name	chkCo
	Caption	Квадрат коэффициента корреляции

Для завершения создания проекта в модуле рабочего листа **Итого** добавьте код (листинг 8.5, *а*) приводимых ниже трех процедур обработки событий `Change` каждого из флажков, где:

- флажок **Тренд** обеспечивает построение или удаление линии тренда. Если линия тренда удалена, то флажки **Уравнение** и **Квадрат коэффициента корреляции** блокируются;
- флажок **Уравнение** отвечает за вывод уравнения на линию тренда;
- флажок **Квадрат коэффициента корреляции** управляет отображением значения одноименного коэффициента.

Листинг 8.5, *а*. Линия тренда. Модуль рабочего листа Итого

```
Private Sub chkTrend_Change()
   On Error Resume Next
   If lstType.Text = xlPie Then Exit Sub
   ActiveSheet.ChartObjects(1).Activate
   Dim c As Chart
   Set c = ActiveChart
   If chkTrend.Value Then
      chkEq.Enabled = True
      chkCo.Enabled = True
      c.SeriesCollection(1).Trendlines.Add _
           Type:=xlLinear, _
           Forward:=0, _
           Backward:=0, _
           DisplayEquation:=False, _
           DisplayRSquared:=False
      If chkTrend.Value Then
          c.SeriesCollection(1).Trendlines(1). _
             DisplayEquation = True
      End If
      If chkCo.Value Then
          c.SeriesCollection(1).Trendlines(1). _
             DisplayRSquared = True
      End If
   Else
      c.SeriesCollection(1).Trendlines(1).Delete
      chkEq.Enabled = False
      chkCo.Enabled = False
   End If
End Sub
```

```
Private Sub chkEq_Click()
   If chkTrend.Value Then
      Dim c As Chart
      ActiveSheet.ChartObjects(1).Activate
      Set c = ActiveChart
      If chkEq.Value Then
         c.SeriesCollection(1).Trendlines(1). _
            DisplayEquation = True
      Else
         c.SeriesCollection(1).Trendlines(1). _
            DisplayEquation = False
      End If
   End If
End Sub

Private Sub chkCo_Click()
   If chkTrend.Value Then
      Dim c As Chart
      ActiveSheet.ChartObjects(1).Activate
      Set c = ActiveChart
      If chkCo.Value Then
        c.SeriesCollection(1).Trendlines(1). _
           DisplayRSquared = True
      Else
        c.SeriesCollection(1).Trendlines(1). _
           DisplayRSquared = False
      End If
   End If
End Sub
```

В модуль `ЭтаКнига` добавьте следующую процедуру, устанавливающую начальные состояния флажков (листинг 8.5, *б*). Конечно, вызов процедуры `InitTrend` надо также добавить в процедуру `Workbook_Open`, расположенную в том же модуле.

Листинг 8.5, *б*. Линия тренда. Модуль `ЭтаКнига`

```
Private Sub InitTrend()
   With Worksheets("Итого")
      .chkTrend.Value = False
      .chkEq.Value = False
      .chkCo.Value = False
   End With
End Sub
```

Построение поверхности и управление ее ориентацией

В предыдущих разделах была построена итоговая таблица расходов фирмы ООО "Бескрайние просторы" и рассмотрены различные способы графического представления данных, содержащихся в этой таблице. Все построенные диаграммы были двухмерные. В данном разделе мы представим данные из итоговой таблицы в виде поверхности, по оси абсцисс которой откладываются типы расходов, ординат — месяцы, а оси Z — расходы. Поверхность будет строиться автоматически по таблице при открытии рабочей книги. Наглядность представления данных в виде поверхности существенно зависит от того, с какой стороны и под каким углом пользователь на нее смотрит. Поэтому дополнительно к программированию процесса создания поверхности расположим на рабочем листе два элемента управления — два списка. Первый список предназначен для изменения угла возвышения, с которого мы смотрим на поверхность. Установите значение его свойства `Name` равным `lstElevation`. Второй — для поворота поверхности вокруг оси Z. Установите значение его свойства `Name` равным `lstRotation` (рис. 8.11). В списки при активизации рабочего листа выводятся до-

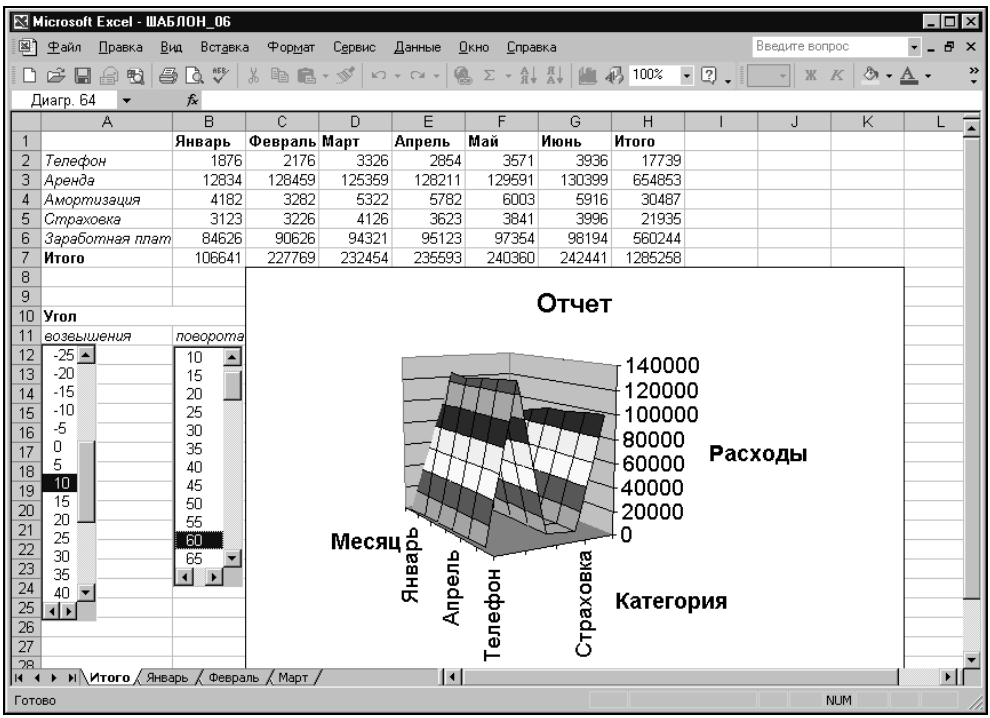


Рис. 8.11. Вращение поверхности

пустимые углы. Для завершения проекта наберите в стандартном модуле ЭтаКнига и модуле рабочего листа **Итого** код приводимых ниже процедур (листинг 8.6).

Листинг 8.6, *а*. Вращение диаграммы. Модуль ЭтаКнига

```
Private Sub Workbook_Open()
   Dim i As Integer
   With Worksheets("Итого")
      If .ChartObjects.Count > 0 Then
          .ChartObjects.Delete
      End If
     Chart3DBuilder
     With .lstElevation
         .Clear
         For i = -90 To 90 Step 5
            .AddItem i
         Next
        .ListIndex = 21
      End With
      With .lstRotation
         .Clear
         For i = 0 To 360 Step 5
            .AddItem i
         Next
         .ListIndex = 4
      End With
   End With
End Sub

Private Sub Chart3DBuilder()
    Dim r As Range
    Set r = Sheets("Итого").Range("A1").CurrentRegion
    Set r = r.Resize(r.Rows.Count - 1, r.Columns.Count - 1)
    r.Select
    Charts.Add
    ActiveChart.ChartType = xlSurface
    ActiveChart.SetSourceData Source:=r, PlotBy:=xlRows
    ActiveChart.Location Where:=xlLocationAsObject, Name:="Итого"
    With ActiveChart
        .HasTitle = True
        .ChartTitle.Characters.Text = "Отчет"
        .Axes(xlCategory).HasTitle = True
```

```
        .Axes(xlCategory).AxisTitle.Characters.Text = "Месяц"
        .Axes(xlSeries).HasTitle = True
        .Axes(xlSeries).AxisTitle.Characters.Text = "Категория"
        .Axes(xlValue).HasTitle = True
        .Axes(xlValue).AxisTitle.Characters.Text = "Расходы"
    End With
    ActiveChart.HasLegend = False
    With Worksheets("Итого").ChartObjects(1)
       .Top = Range("C8").Top
       .Left = Range("C8").Left
       .Width = Range("C8:K29").Width
       .Height = Range("C8:K29").Height
    End With
End Sub
```

Листинг 8.6, *б*. Вращение диаграммы. Модуль рабочего листа

```
Private Sub lstElevation_Click()
   On Error Resume Next
   ChartRotator lstRotation.Value, lstElevation.Value
End Sub

Private Sub lstRotation_Click()
   ChartRotator lstRotation.Value, lstElevation.Value
End Sub

Private Sub ChartRotator(valRotation As Integer, _
                         valElevation As Integer)
   If ActiveSheet.ChartObjects.Count <> 1 Then
      Exit Sub
   End If
   ActiveSheet.ChartObjects(1).Activate
   If ActiveChart.ChartType <> xlSurface Then
      Exit Sub
   End If
   With ActiveChart
      .Elevation = valElevation
      .Rotation = valRotation
   End With
End Sub
```

Установка защиты на вложенную в рабочий лист диаграмму

Для установки защиты на вложенную в рабочий лист диаграмму можно воспользоваться методом `Password` объекта `Worksheets` со значением параметра `UserInterfaceOnly` равным `True`, что обеспечит защиту рабочего листа только от действий пользователя, в то же время разрешая ввод данных в ячейки из кода. Например, в следующем коде (листинг 8.7) устанавливается защита на все объекты рабочего листа **Итого** с отчетной таблицей, а пользователю разрешается ввод данных только в диапазон **B2:G6** (рис. 8.12).

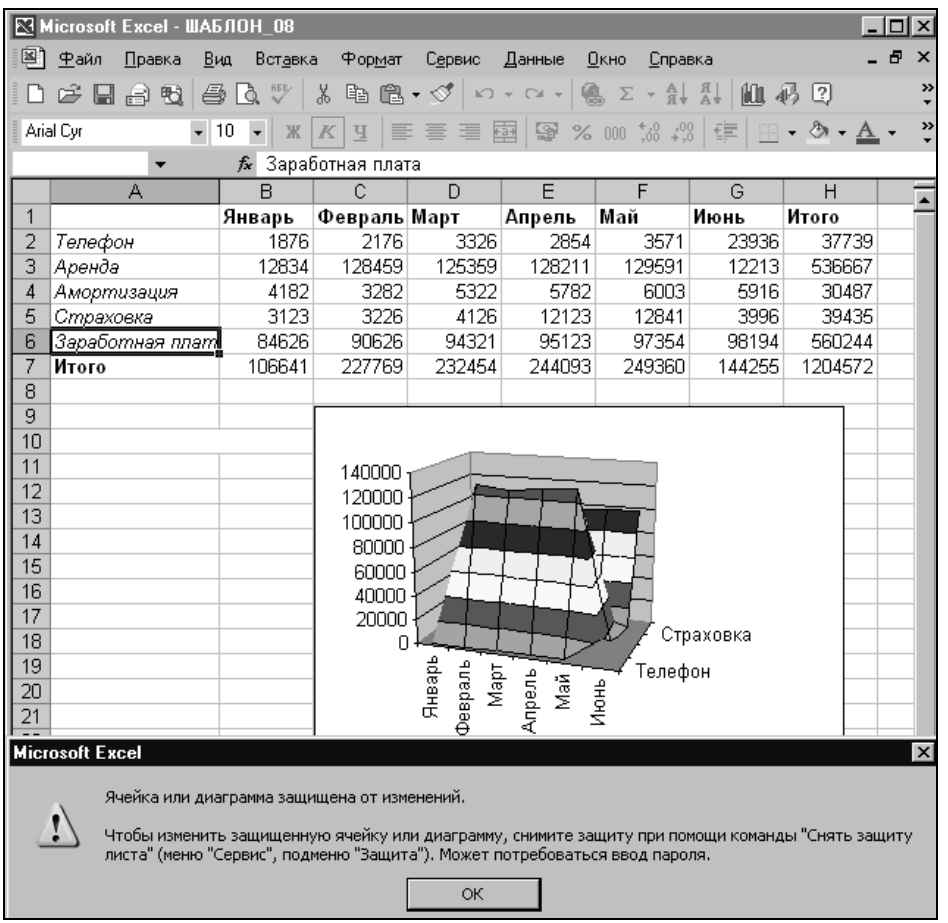


Рис. 8.12. Сообщение, отображаемое при попытке ввести данные в ячейки вне диапазона **B2:G6**

Листинг 8.7. Установка защиты на внедренную диаграмму. Модуль ЭтаКнига

```
Private Sub Workbook_Open()
   SetPtotection
End Sub

Private Sub SetPtotection()
   On Error Resume Next
   Worksheets("Итоги").Range("B2:G6").Locked = False
   Worksheets("Итоги").Protect Password:="1", UserInterfaceOnly:=True
End Sub
```

Защита диаграммы, расположенной на отдельном листе

У диаграммы, расположенной на отдельном листе, т. е. у объекта `Chart`, имеется метод `Protect`, позволяющий установить защиту на диаграмму.

```
Protect(Password, DrawingObjects, Contents, Scenarios, UserInterfaceOnly)
```

- *Password* — задает пароль защиты;
- *DrawingObjects* — устанавливает защиту на графические объекты;
- *Contents* — определяет защиту всей диаграммы;
- *Scenarios* — задает защиту на сценарии;
- *UserInterfaceOnly* — защита пользовательского интерфейса, но не макросов. Если этот параметр опущен, то защита применяется как к интерфейсу, так и макросам.

Защита снимается методом `Unprotect`.

```
Unprotect(Password)
```

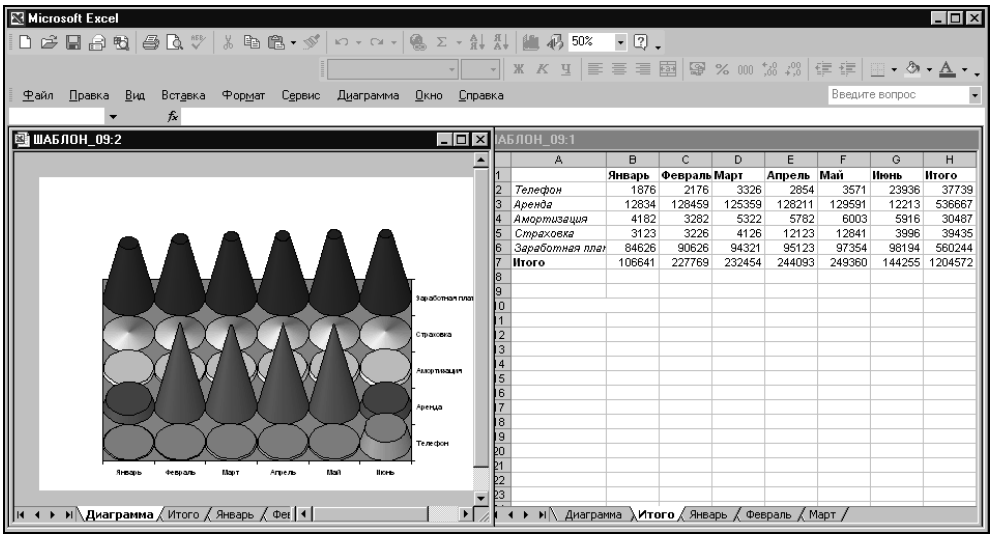
Например, следующий код (листинг 8.8) устанавливает защиту диаграммы, расположенной на отдельном листе, а также блокировку для пользователя всех ячеек рабочего листа **Итого**, за исключением диапазона **B2:G6** (рис. 8.13).

Листинг 8.8. Установка защиты на диаграмму. Модуль ЭтаКнига

```
Private Sub Workbook_Open()
   SetPtotection
End Sub

Private Sub SetPtotection()
    On Error Resume Next
```

```
    Charts(1).Protect Password:="d1", UserInterfaceOnly:=True
    Worksheets("Итоги").Range("B2:G6").Locked = False
    Worksheets("Итоги").Protect Password:="1"
End Sub
```



	A	B	C	D	E	F	G	H
1		Январь	Февраль	Март	Апрель	Май	Июнь	Итого
2	*Телефон*	1876	2176	3326	2854	3571	23936	37739
3	*Аренда*	12834	128459	125359	128211	129591	12213	536667
4	*Амортизация*	4182	3282	5322	5782	6003	5916	30487
5	*Страховка*	3123	3226	4126	12123	12841	3996	39435
6	*Заработная плат*	84626	90626	94321	95123	97354	98194	560244
7	**Итого**	106641	227769	232454	244093	249360	144255	1204572

Рис. 8.13. Установка защиты на диаграмму

События и диаграммы

События по умолчанию ассоциируются с диаграммами, созданными на отдельных листах диаграмм. Например, следующий код из модуля диаграммы (листинг 8.9) обрабатывает событие `MouseDown` и в зависимости от местоположения щелчка мышью изменяет цвет как области, так и самой диаграммы.

Листинг 8.9. Изменение цвета диаграммы в зависимости от местоположения щелчка мышью. Код из модуля диаграммы

```
Private Sub Chart_MouseDown(ByVal Button As Long, ByVal Shift As Long, _
     ByVal x As Long, ByVal y As Long)
   If x < ActiveWindow.UsableWidth / 2 Then
      ActiveChart.SeriesCollection(1).Interior.Color = RGB(255, 0, 0)
      ActiveChart.ChartArea.Interior.Color = RGB(255, 255, 0)
   Else
      ActiveChart.SeriesCollection(1).Interior.Color = RGB(255, 255, 0)
```

```
      ActiveChart.ChartArea.Interior.Color = RGB(255, 0, 0)
   End If
End Sub
```

Привязка событий к вложенным в рабочий лист диаграммам

События со вложенными в рабочий лист диаграммами явным образом не ассоциированы. Для того чтобы их все-таки можно было обработать, надо проделать следующие подготовительные процедуры:

1. Создать модуль класса и назвать его, например, `MyEventClassModule`.
2. Объявить с ключевым словом `WithEvents` переменную типа `Chart` в модуле класса `MyEventClassModule` (листинг 8.10, *а*). После этого в редакторе кода модуля класса в списке объектов появится объект `MyEventClassModule`, а в списке событий — все связанные с диаграммами события.

Листинг 8.10, *а*. Привязка событий к вложенным в рабочий лист диаграммам. Модуль класса `MyEventClassModule` (вариант 1)

```
Public WithEvents MyChartClass As Chart
```

3. В редакторе кода набрать код обработки тех событий, которых требует бизнес-логика вашего проекта. Например, в данном случае (листинг 8.10, *б*) при нажатии кнопки мыши отобразится предупреждающее сообщение "`Не двигайте ее`".

Листинг 8.10, *б*. Привязка событий к вложенным в рабочий лист диаграммам. Модуль класса `MyEventClassModule` (вариант 2)

```
Private Sub MyChartClass_MouseDown(ByVal Button As Long, _
      ByVal Shift As Long, ByVal x As Long, ByVal y As Long)
    MsgBox "Не двигайте ее"
End Sub
```

4. Остается только где-нибудь указать, что объект `MyChartClass` указывает на искомую вложенную диаграмму. Например, это можно сделать на этапе открытия книги, добавив в модуль `ЭтаКнига` следующий код (листинг 8.10, *в*), который ассоциирует первую из вложенных в первый рабочий лист диаграмм с объектом `MyChartClass`. Теперь при нажатии кнопки мыши на этой диаграмме отобразится сообщение "`Не двигайте ее`".

Листинг 8.10, *в*. Привязка событий к вложенным в рабочий лист диаграммам. Модуль ЭтаКнига

```
Dim MyClassModule As New MyEventClassModule

Sub ChartInit()
   Set MyClassModule.MyChartClass = Worksheets(1).ChartObjects(1).Chart
End Sub

Private Sub Workbook_Open()
   ChartInit
End Sub
```

Изменение типа диаграммы при помощи контекстного меню

В качестве примера использования событий диаграмм приведем следующий проект, в котором при щелчке правой кнопкой мыши на диаграмме отображается контекстное меню, состоящее из трех команд: **Гистограмма**, **График** и **Круговая**, выбор которых и приводит к изменению типа диаграммы (рис. 8.14).

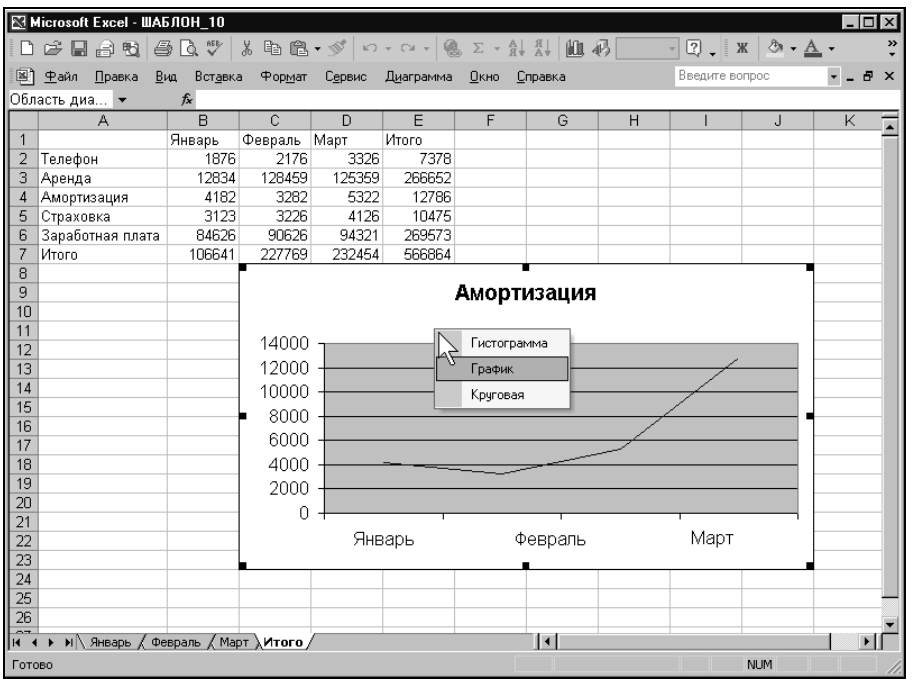


Рис. 8.14. Изменение типа диаграммы при помощи контекстного меню

Так как обрабатывается событие "щелчок правой кнопкой на внедренной диаграмме", то необходимо создать класс (в данном случае `ChartEventClass`), в котором и реализуется обрабатывающий код. При открытии рабочей книги надо связать экземпляр класса `ChartEventClass` с конкретной диаграммой, создать контекстное меню, а также назначить командам этого меню соответствующие макросы, которые и осуществляют изменение типа диаграммы (листинг 8.11).

Листинг 8.11, *а*. Изменение типа диаграммы. Модуль `ЭтаКнига`

```
Dim ChartEvent As New ChartEventClass

Private Sub Workbook_Open()
   TbInit
   ChartInit
   PopUpBuilder
End Sub

Sub ChartInit()
   Set ChartEvent.ChartClass = Worksheets("Итого").ChartObjects(1).Chart
End Sub

Sub TbInit()
   tb(0, 0) = "Гистограмма" : tb(0, 1) = xlColumnClustered
   tb(1, 0) = "График" : tb(1, 1) = xlLine
   tb(2, 0) = "Круговая" : tb(2, 1) = xlPie
End Sub

Private Sub PopUpBuilder()
   Dim btnGraph(2) As CommandBarButton
   Set cbPopUp = Application.CommandBars.Add(Name:="CustomBarPopup", _
                                             Position:=msoBarPopup)
   Dim i As Integer
   For i = 0 To 2
      With cbPopUp.Controls
         Set btnGraph(i) = .Add(Type:=msoControlButton)
         With btnGraph(i)
           .Caption = tb(i, 0)
           .Style = msoButtonCaption
           .OnAction = "Press" & i
         End With
      End With
   Next
End Sub
```

```
Private Sub Workbook_BeforeClose(Cancel As Boolean)
   Application.CommandBars("CustomBarPopup").Delete
End Sub
```

Листинг 8.11, *б*. Изменение типа диаграммы. Стандартный модуль

```
Public cbPopUp As CommandBar
Public tb(2, 1) As Variant

Sub Press0()
    ChangeTypeChart 0
End Sub

Sub Press1()
   ChangeTypeChart 1
End Sub

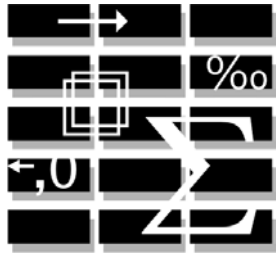
Sub Press2()
   ChangeTypeChart 2
End Sub

Sub ChangeTypeChart(idx As Integer)
   ActiveSheet.ChartObjects(1).Activate
   ActiveChart.ChartType = tb(idx, 1)
   If tb(idx, 1) = xlPie Then
      ActiveChart.HasLegend = True
   Else
      ActiveChart.HasLegend = False
   End If
End Sub
```

Листинг 8.11, *в*. Изменение типа диаграммы. Модуль класса `ChartEventClass`

```
Public WithEvents ChartClass As Chart

Private Sub ChartClass_BeforeRightClick(Cancel As Boolean)
   Cancel = True
   cbPopUp.ShowPopup
End Sub
```



ГЛАВА 9

Создание меню, контекстного меню и панели инструментов

В предыдущих главах было приведено огромное количество примеров создания приложений, позволяющих автоматизировать и существенно упростить работу в MS Excel. Одной из наиболее важных задач при разработке приложения является создание собственного интерфейса, при этом необходимо исключить ненужные панели инструментов, создать пользовательские панели инструментов и преобразовать строку меню под нужды приложения. Как раз одним из достоинств MS Excel является то, что его интерфейс легко настроить под конкретное приложение, создавая и настраивая панели инструментов и строки меню. Панели инструментов, строки меню и контекстные меню (вызываемые щелчком правой кнопки мыши) фактически являются одинаковыми объектами, для которых в интерфейсе пользователя используется термин *панели инструментов*, а в VBA — термин *панели команд* (command bar). Данные о них инкапсулируются в объектах `CommandBar`, которые в своей совокупности образуют семейство `CommandBars`.

Семейство *CommandBars*

Семейство `CommandBars` состоит из объектов `CommandBar`, которые инкапсулируют в себе данные о меню, контекстном меню или панели инструментов. В этом семействе имеется единственное событие `OnUpdate`, которое генерируется при изменениях в каком-либо из элементов семейства.

```
Private Sub CommandBars_OnUpdate()
```

Кроме того, семейство `CommandBars` обладает методами и свойствами (табл. 9.1 и 9.2), позволяющими добавлять и находить элемент семейства, а также контролировать внешний вид элементов, например, отображать крупные значки или оставить размеры, устанавливаемые по умолчанию.

Таблица 9.1. *Методы семейства* `CommandBars`

Метод	Описание
`Add`	Добавляет новый элемент в семейство. `Add(`*`Name`*`,` *`Position`*`,` *`MenuBar`*`,` *`Temporary`*`)` ♦ *`Name`* — имя создаваемого меню или панели инструментов; ♦ *`Position`* — определяет местоположение панели команд. Допустимыми значения являются следующие константы `MsoBarPosition`: `msoBarLeft` (привязка строки меню к левой границе окна основного приложения), `msoBarTop` (привязка строки меню к верхней границе окна основного приложения), `msoBarRight` (привязка строки меню к правой границе окна основного приложения), `msoBarBottom` (привязка строки меню к нижней границе окна основного приложения), `msoBarFloating` (свободное размещение строки меню на экране), `msoBarPopup` (устанавливает, будет ли панель команд контекстным меню); ♦ *`MenuBar`* — параметр, принимающий логические значения. Если его значение равно `True`, то панель команд заменяет активную строку меню; ♦ *`Temporary`* — параметр, принимающий логические значения. Если его значение равно `True`, то панель команд удаляется по закрытию MS Excel, но *не по закрытию книги*
`FindControl`	Возвращает объект `CommandBarControl`, удовлетворяющий указанным критериям
`FindControls`	Возвращает семейство `CommandBarControls`, удовлетворяющее указанным критериям
`ReleaseFocus`	Переводит фокус с меню или панели инструментов

Таблица 9.2. *Свойства семейства* `CommandBars`

Свойство	Описание
`Count`	Число элементов семейства
`DisableCustomize`	Определяет, можно ли производить настройку меню
`DisplayFonts`	Определяет, отображать ли текст указанным шрифтом в меню шрифта
`DisplayTooltips`	Определяет, отображать ли всплывающие подсказки, когда указатель мыши находится над элементом меню
`LargeButtons`	Определяет, имеют ли кнопки увеличенный размер

Объект *CommandBar*

Данные о меню, контекстном меню или панели инструментов инкапсулируются в объекте `CommandBar`. Кроме того, объект `CommandBar` обладает методами и свойствами, перечисленными в табл. 9.3 и 9.4, которые позволяют удалять, переустанавливать параметры объекта, а также контролировать его размеры, местоположение, видимость и защиту.

Таблица 9.3. *Методы объекта* `CommandBar`

Метод	Описание
`Delete`	Удаляет меню или панель инструментов
`FindControl`	Возвращает объект `CommandBarControl`, удовлетворяющий указанным критериям
`Reset`	Восстанавливает в исходное состояние (по умолчанию) встроенные меню или панели инструментов
`ShowPopUp`	Отображает контекстную панель команд в специфицированном местоположении, если указаны координаты, либо в месте расположения указателя, если они опущены. `ShowPopUp(x, y)` Здесь `x`, `y` — x- и y-координаты панели команд. Если какой-то из параметров опущен, то используется соответствующая текущая координата курсора

Таблица 9.4. *Свойства объекта* `CommandBar`

Свойство	Описание
`BuiltIn`	Определяет, является ли меню или панель инструментов встроенными
`Controls`	Возвращает семейство `CommandBarControls`, элементами которого являются пункты меню или элементы управления панели инструментов
`Enabled`	Определяет, возможен ли доступ к меню или панели инструментов
`Height`, `Width`	Высота и ширина меню или панели инструментов
`Id`	Идентификатор меню или панели инструментов
`Index`	Индекс меню или панели инструментов в семействе `CommandBars`
`Left`, `Top`	Координаты левого верхнего угла меню или панели инструментов

Таблица 9.4 *(окончание)*

Свойство	Описание
Name	Имя меню или панели инструментов
Position	Возвращает местоположение панели команд. Допустимыми значениями являются следующие константы MsoBarPosition: msoBarLeft, msoBarTop, msoBarRight, msoBarBottom, msoBarFloating, msoBarPopup
Protection	Защита меню или панели инструментов. Допустимыми значениями являются следующие константы MsoBarProtection: msoBarNoProtection, msoBarNoCustomize, msoBarNoResize, msoBarNoMove, msoBarNoChangeVisible, msoBarNoChangeDock, msoBarNoVerticalDock, msoBarNoHorizontalDock. Если необходимо использовать защиту по нескольким позициям, то значение свойства Protection надо установить как сумму соответствующих констант
Visible	Определяет, отображено или скрыто меню или панель инструментов

Встроенные панели инструментов

Как отмечалось ранее, в семействе CommandBars хранятся все строки меню и панели инструментов конкретного приложения. В табл. 9.5 приведены названия наиболее часто используемых встроенных панелей инструментов MS Excel.

Таблица 9.5. *Названия встроенных панелей инструментов*

Название (англ.)	Название (рус.)
Formatting	Форматирования
Standard	Стандартная
Visual Basic	Visual Basic
Drawing	Рисование
Web	Web
WordArt	WordArt
External Data	Внешние данные
Chart	Диаграммы
Picture	Настройка изображения

Таблица 9.5 *(окончание)*

Название (англ.)	Название (рус.)
Worksheet Menu Bar	Строка меню листа
Reviewing	Рецензирование
PivotTable	Сводные таблицы
Control Toolbox	Элементы управления
Forms	Формы
Clipboard	Буфер обмена
Stop Recording	Остановка записи
Shadow Settings	Настройка тени
3-D Settings	Настройка объема
Circular Reference	Циклические ссылки
Full Screen	Во весь экран
Chart Menu Bar	Строка меню диаграммы
Auditing	Зависимости

Пункты меню или элементы управления панели инструментов и семейство *CommandBarControls*

Свойство `Controls` объекта `CommandBar` возвращает семейство `CommandBarControls` объектов `CommandBarControl`, которые инкапсулируют в себе данные о пунктах меню или элементов управления панели инструментов.

Элементы семейства `CommandBarControls` относятся к одному из трех типов, перечисленных в табл. 9.6.

Таблица 9.6. *Элементы семейства* `CommandBarControls`

Элемент	Описание
`CommandBarButton`	Кнопка или элемент меню, вызывающие выполнение команды или подпрограммы
`CommandBarComboBox`	Сложно организованные меню, такие как поле ввода, раскрывающийся список или поле со списком
`CommandBarPopUp`	Меню или вложенное меню

В семействе CommandBarControls основным свойством является свойство Count, возвращающее число элементов семейства, и метод Add, добавляющий в семейство новый элемент.

Add(*Type*, *Id*, *Parameter*, *Before*, *Temporary*)

- *Type* — задает тип добавляемого элемента. Допустимыми значениями являются следующие константы MsoControlType: msoControlButton (кнопка или элемент меню), msoControlEdit (поле ввода), msoControlDropDown (раскрывающийся список), msoControlComboBox (комбинированное поле) и msoControlPopup (вложенное меню);
- *Id* — целое число, идентифицирующее встроенный элемент управления. Если *Id* равно 1 или опущено, то добавляется пустой элемент управления заданного типа;
- *Parameter* — задает параметр, используемый встроенным элементом управления;
- *Before* — индекс или имя элемента управления, перед которым добавляется новый элемент управления. Если аргумент опущен, то элемент размещается в конце панели инструментов или меню;
- *Temporary* — параметр, принимающий логические значения. Если его значение равно True, то новый элемент управления автоматически удаляется из меню или панели инструментов при закрытии приложения.

Пункт меню или кнопка панели инструментов и объект *CommandBarButton*

Объект CommandBarButton инкапсулирует в себе данные о пункте меню или кнопке панели инструментов. В табл. 9.7 и 9.8 перечислены свойства и методы этого объекта, определяющие внешний вид кнопки (текст, рисунок, размеры) и обеспечивающие широкий спектр воздействия на объект от вставки в него изображения до выполнения процедуры, ассоциированной с объектом.

***Таблица 9.7.** Свойства объекта* CommandBarButton

Свойство	Описание
BeginGroup	Проверяет, является ли пункт меню или кнопка первыми в группе
BuiltIn	Проверяет, является ли пункт меню или кнопка встроенными
BuiltInFace	Проверяет, имеет ли кнопка встроенное изображение

Таблица 9.7 *(продолжение)*

Свойство	Описание
`Caption`	Текст, отображаемый в пункте меню или кнопке
`DescriptionText`	Описание элемента управления. Используется при составлении документации
`Enabled`	Устанавливает, доступен ли пользователю выбор элемента управления
`FaceID`	Возвращает идентификатор встроенной кнопки, используемой в качестве элемента управления. Если это свойство равно 0, то возвращается пустой элемент управления. Значение `2950` соответствует пользовательской кнопке 
`Height`, `Width`	Высота и ширина пункта меню или кнопки
`Id`	Идентификатор пункта меню или кнопки. Для пользовательских пунктов меню или кнопок его значение равно 1
`Index`	Индекс объекта в семействе `CommandBarControls`
`Mask`	Возвращает объект `IPictureDisp`, который представляет собой прозрачную часть изображения меню или кнопки
`OnAction`	Возвращает имя макроса или процедуры, выполняемой при выборе элемента управления
`Picture`	Возвращает объект `IPictureDisp`, который представляет собой изображение, отображаемое в элементе управления
`ShortcutText`	Возвращает комбинацию "горячих" клавиш, назначенных элементу управления
`State`	Возвращает состояние кнопки. Допустимыми значениями являются следующие константы `MsoButtonState`: `msoButtonDown`, `msoButtonMixed`, `msoButtonUp`.
`Style`	Устанавливает внешний вид кнопки. Допустимыми значениями являются следующие константы `MsoButtonStyle`: `msoButtonAutomatic` (кнопка без значка и текста), `msoButtonIcon` (кнопка со значком, в этом случае надо также задать свойство `Icon`), `msoButtonCaption` (кнопка с надписью, в данном случае надо также задать свойство `Caption`), `msoButtonIconAndCaption` (кнопка со значком и надписью, в этом случае надо также задать свойства `Icon` и `Caption`), `msoButtonIconAndCaptionBelow` (кнопка с надписью, а затем со значком), `msoButtonIconAndWrapCaption` (кнопка со значком и многострочной надписью), `msoButtonIconAndWrapCaptionBelow` (кнопка с многострочной надписью, а затем со значком), `msoButtonWrapCaption` (кнопка с многострочной надписью)

Таблица 9.7 *(окончание)*

Свойство	Описание
`Tag`	Служебная информация для разработчика
`TooltipText`	Текст, отображаемый в окне всплывающей подсказки
`Visible`	Устанавливает, отображается ли на экране кнопка или пункт меню

Таблица 9.8. *Методы объекта* `CommandBarButton`

Метод	Описание
`Copy`	Копирует существующий пункт меню или кнопку в меню или панель инструментов
`CopyFace`	Копирует изображение кнопки в буфер обмена
`Delete`	Удаляет пункт меню или кнопку
`Execute`	Выполняет процедуру, назначенную пункту меню или кнопке
`Move`	Перемещает пункт меню или кнопку
`PasteFace`	Вставляет изображение из буфера обмена на поверхность кнопки
`Reset`	Устанавливает назначенные по умолчанию параметры пункту меню или кнопке встроенного меню или панели инструментов
`SetFocus`	Устанавливает фокус на пункте меню или кнопке

Создание строки меню

В качестве примера создадим строку меню, имеющую простую структуру: два выпадающих меню — **Файл** и **Справка**. Выпадающее меню **Файл** включает три пункта меню: **Создать**, **Сохранить** и **Выход**. В выпадающем меню **Справка** имеется единственный пункт меню — **Справка**. При выборе каждой из команд меню, за исключением **Выход**, для подтверждения действий пользователя выполняются *процедуры-заглушки*, которые лишь обозначают присутствие кода. Выбор же команды **Выход** действительно производит закрытие приложения (рис. 9.1).

В данном проекте код набирается в двух модулях: `ЭтаКнига` и стандартном.

В модуле `ЭтаКнига` (листинг 9.1, *а*) имеются две процедуры. Первая из них обрабатывает событие `Open` объекта `Workbook`. Эта процедура вызывает процедуру `MenuBuilder`, в которую помещен весь код, и создает строку меню,

которая в коде получает имя `МояСтрокаМеню`. Для того чтобы методом `Add` семейства `CommandBars` была создана строка меню, а не панель инструментов, значение параметра `MenuBar` должно быть установлено равным `True`. Так как значение параметра `Temporary` метода `Add` установлено равным `True`, то строка меню является временной и удаляется при закрытии всего приложения. Но это не всегда удобно. Лучше бы было, если бы строка меню удалялась при закрытии книги. Для этого в коде и обрабатывается событие `BeforeClose` объекта `Workbook`. В процедуре обработки этого события как раз и удаляется пользовательская строка меню.

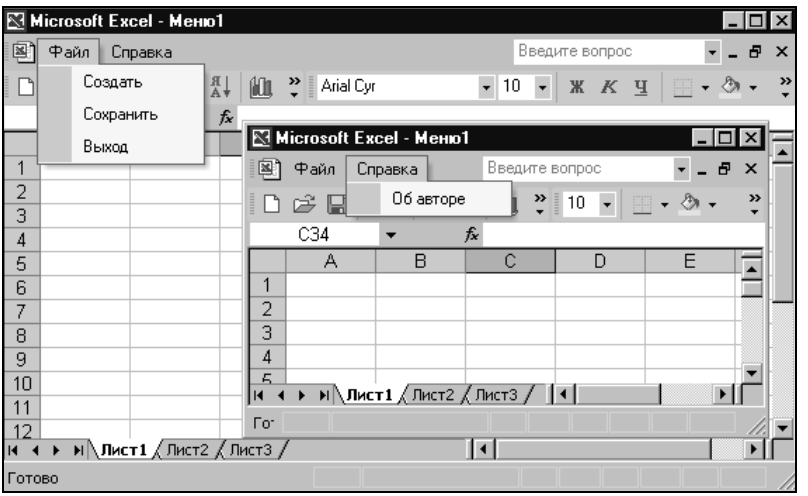


Рис. 9.1. Строка меню

В стандартном модуле (листинг 9.1, *б*) расположен ряд процедур, которые назначены соответствующим пунктам меню при помощи свойства `OnAction`.

Листинг 9.1, *а*. Строка меню. Модуль `ЭтаКнига`

```
Private Sub Workbook_Open()
   MenuBuilder
End Sub

Private Sub MenuBuilder()
   Dim cbMenuBar As CommandBar
   Set cbMenuBar = Application.CommandBars.Add(Name:="МояСтрокаМеню", _
                                               Position:=msoBarTop, _
                                               MenuBar:=True, _
                                               Temporary:=True)
   With cbMenuBar
      .Visible = True
```

```
      With .Controls
         With .Add(Type:=msoControlPopup)
            .Caption = "Файл"
            With .Controls
               With .Add(Type:=msoControlButton)
                   .Caption = "Создать"
                   .OnAction = "NewDoc"
               End With
               With .Add(Type:=msoControlButton)
                  .Caption = "Сохранить"
                  .OnAction = "SaveDoc"
               End With
               With .Add(Type:=msoControlButton)
                  .Caption = "Выход"
                  .OnAction = "ExitApp"
               End With
            End With
         End With
         With .Add(Type:=msoControlPopup)
         .Caption = "Справка"
            With .Controls
               With .Add(Type:=msoControlButton)
                  .Caption = "Об авторе"
                  .OnAction = "AboutAuthor"
               End With
            End With
         End With
      End With
   End With
End Sub

Private Sub Workbook_BeforeClose(Cancel As Boolean)
   Application.CommandBars("МояСтрокаМеню").Delete
End Sub
```

Листинг 9.1, *б*. Строка меню. Стандартный модуль

```
Public Sub NewDoc()
 MsgBox "New"
End Sub

Public Sub SaveDoc()
 MsgBox "Save"
End Sub
```

```
Public Sub ExitApp()
 Application.Quit
End Sub

Public Sub AboutAuthor()
 MsgBox "James Bond", vbInformation, "About Author"
End Sub
```

Создание сложного меню

Пунктами меню могут быть выпадающие меню, что позволяет создавать меню произвольной конфигурации. Например, создадим меню, показанное на рис. 9.2. В выпадающем меню **Операции** имеются два пункта: **Цвет** и **Выход**, причем пункт **Цвет** является, в свою очередь, выпадающим меню, состоящим из двух пунктов: **Красный** и **Желтый**. При выборе команды **Красный** или **Желтый** производится окраска активного рабочего листа в соответствующий цвет, а при выборе команды **Выход** — закрывается рабочая книга.

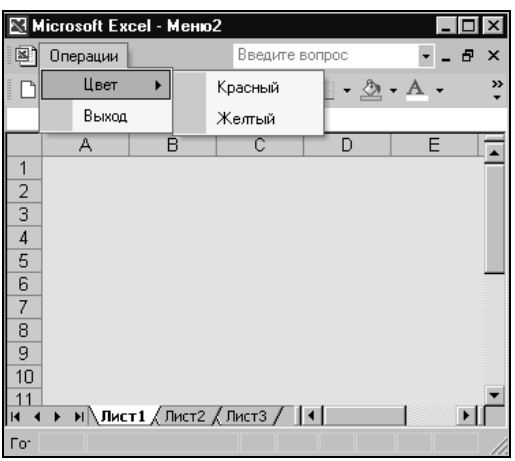


Рис. 9.2. Сложное меню

Добавление в меню не пункта, а выпадающего меню производится установкой у метода `Add` семейства `Controls` значения параметра `Type` равным `msoControlPopup`. Итак, для реализации данного проекта остается в модулях `ЭтаКнига` и стандартном набрать приводимый ниже код (листинг 9.2).

Листинг 9.2, *а*. Сложное меню. Модуль `ЭтаКнига`

```
Private Sub Workbook_Open()
   MenuBuilder
End Sub
```

```
Private Sub MenuBuilder()
   Dim cbMenuBar As CommandBar
   Dim btnOperation As CommandBarPopup
   Dim btnColor As CommandBarPopup
   Dim btnRed As CommandBarButton
   Dim btnYellow As CommandBarButton
   Dim btnExit As CommandBarButton

   Set cbMenuBar = Application.CommandBars.Add(Name:="МоеСложноеМеню", _
                                               Position:=msoBarTop, _
                                               MenuBar:=True, _
                                               Temporary:=True)
   With cbMenuBar
      .Visible = True
      With .Controls
         Set btnOperation = .Add(Type:=msoControlPopup)
      End With
   End With

   With btnOperation
       .Caption = "Операции"
       With .Controls
          Set btnColor = .Add(Type:=msoControlPopup)
          Set btnExit = .Add(Type:=msoControlButton)
       End With
   End With

   With btnColor
      .Caption = "Цвет"
      With .Controls
         Set btnRed = .Add(Type:=msoControlButton)
         Set btnYellow = .Add(Type:=msoControlButton)
      End With
   End With

   With btnRed
      .Caption = "Красный"
      .OnAction = "FillRed"
   End With
   With btnYellow
      .Caption = "Желтый"
      .OnAction = "FillYellow"
   End With
   With btnExit
      .Caption = "Выход"
```

```
      .OnAction = "ExitApp"
   End With
End Sub

Private Sub Workbook_BeforeClose(Cancel As Boolean)
   Application.CommandBars("МоеСложноеМеню").Delete
End Sub
```

Листинг 9.2, *б*. Сложное меню. Стандартный модуль

```
Public Sub FillRed()
   ActiveSheet.Cells.Interior.Color = RGB(255, 0, 0)
End Sub

Public Sub FillYellow()
   ActiveSheet.Cells.Interior.Color = RGB(255, 255, 0)
End Sub

Public Sub ExitApp()
   ThisWorkbook.Close
End Sub
```

Пункты меню с картинками и со встроенными функциями

В меню можно включать пункты из уже существующих меню со встроенными функциями. Например, если в меню должна быть команда **Сохранить**, почему бы ее не позаимствовать из панели инструментов **Стандартная**. Как это делается? Да, очень просто. Надо определить идентификатор этой команды свойством `ID`, а уже затем определенный пункт меню (в данном случае, **Сохранить**) добавить в строку меню методом `Add` семейства `Controls`, у которого в качестве значения параметра `ID` указано значение идентификатора данной команды.

Если же требуется в пункт меню вставить только картинку из стандартного пункта без инициализации ассоциированных с ним встроенных функций, то в этом случае по идентификатору команды методом `FindControl` семейства `CommandBars` надо получить ссылку на сам объект `CommandBarButton`, далее методом `CopyFace` объекта `CommandBarButton` скопировать рисунок найденной стандартной команды в буфер обмена, а затем уже методом `PasteFace` объекта `CommandBarButton` вставить его в искомую пользовательскую команду.

Для примера создадим еще раз меню из *разд. "Создание строки меню"*. Но в этот раз расположим строку меню вдоль левого края окна приложения

(рис. 9.3). Пункт меню **Сохранить** позаимствуем со всеми встроенными в него функциями из панели инструментов **Стандартная**. В пункты меню **Создать** и **Справка** вставим только картинки из кнопок **Создать** и **Справка: Microsoft Excel** панели инструментов **Стандартная**. Кнопка **Выход** будет теперь обеспечивать закрытие книги без предварительного запроса на сохранения изменений, произведенных в книге (листинг 9.3).

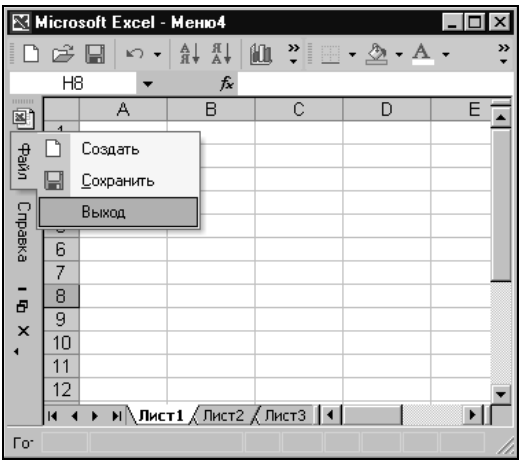


Рис. 9.3. Меню с картинками

Листинг 9.3, *а*. Меню с картинками. Модуль `ЭтаКнига`

```
Private Sub Workbook_Open()
   MenuBuilder
End Sub

Private Sub MenuBuilder()
   Dim cbMenuBar As CommandBar
   Dim btnNew As CommandBarButton
   Dim btnSave As CommandBarButton
   Dim btnExit As CommandBarButton
   Dim btnAbout As CommandBarButton
   Dim n As Integer
   Dim btnTemp As CommandBarButton

   Set cbMenuBar = Application.CommandBars.Add(Name:="МояСтрокаМеню", _
                                               Position:=msoBarLeft, _
                                               MenuBar:=True, _
                                               Temporary:=True)
```

```
With cbMenuBar
   .Visible = True
   With .Controls
      With .Add(Type:=msoControlPopup)
         .Caption = "Файл"
         With .Controls
            Set btnNew = .Add(Type:=msoControlButton)
            n = Application.CommandBars("Standard") _
                .Controls("Сохранить").ID
            Set btnSave = .Add(Type:=msoControlButton, ID:=n)
            Set btnExit = .Add(Type:=msoControlButton)
         End With
      End With
      With .Add(Type:=msoControlPopup)
      .Caption = "Справка"
         With .Controls
            Set btnAbout = .Add(Type:=msoControlButton)
         End With
      End With
   End With
End With

n = Application.CommandBars("Standard").Controls("Создать").ID
Set btnTemp = Application.CommandBars("Standard").FindControl( _
                                  Type:=msoControlButton, ID:=n)
btnTemp.CopyFace
With btnNew
  .PasteFace
  .Caption = "Создать"
  .OnAction = "NewDoc"
End With

With btnExit
   .Caption = "Выход"
   .OnAction = "ExitApp"
End With

n = Application.CommandBars("Standard"). _
        Controls("Справка: Microsoft Excel").ID
Set btnTemp = Application.CommandBars("Standard").FindControl( _
                                    Type:=msoControlButton, ID:=n)
btnTemp.CopyFace
With btnAbout
   .PasteFace
```

```
      .Caption = "Об авторе"
      .OnAction = "AboutAuthor"
   End With
End Sub

Private Sub Workbook_BeforeClose(Cancel As Boolean)
   Application.CommandBars("МояСтрокаМеню").Delete
End Sub
```

Листинг 9.3, *б*. Меню с картинками. Стандартный модуль

```
Public Sub NewDoc()
   MsgBox "New"
End Sub

Public Sub ExitApp()
   ThisWorkbook.Close SaveChanges:=False
End Sub

Public Sub AboutAuthor()
   MsgBox "It's still me", vbInformation
End Sub
```

Скрытие и отображение панелей инструментов

При создании пользовательского интерфейса часто требуется обеспечивать скрытие элементов стандартного интерфейса, например панелей инструментов **Стандартная** и **Форматирование**, за их ненадобностью (рис. 9.4). Подобное скрытие можно реализовать при помощи свойства `Visible` объекта `CommandBar`. А именно при открытии книги установить значения этого свойства для тех панелей инструментов, которые должны быть скрытыми, равным значению `False`, а при закрытии книги — `True`, например, как это делается в следующем коде (листинг 9.4).

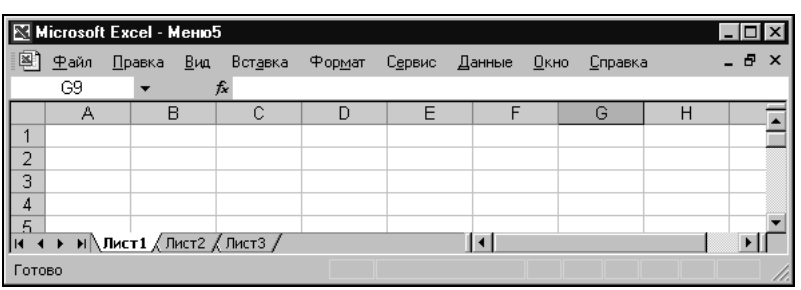


Рис. 9.4. Скрытие панелей инструментов **Стандартная** и **Форматирование**

Листинг 9.4. Скрытие и отображение панелей инструментов. Модуль ЭтаКнига

```
Private Sub Workbook_Open()
   With Application
      .CommandBars("Formatting").Visible = False
      .CommandBars("Standard").Visible = False
   End With
End Sub

Private Sub Workbook_BeforeClose(Cancel As Boolean)
   With Application
      .CommandBars("Formatting").Visible = True
      .CommandBars("Standard").Visible = True
   End With
End Sub
```

Создание панели инструментов с кнопками и защитой от изменений со стороны пользователя

Создадим панель инструментов с тремя кнопками и защитой от изменений со стороны пользователя (рис. 9.5). При этом пользователь не сможет ни перемещать, ни скрывать, ни видоизменять эту панель.

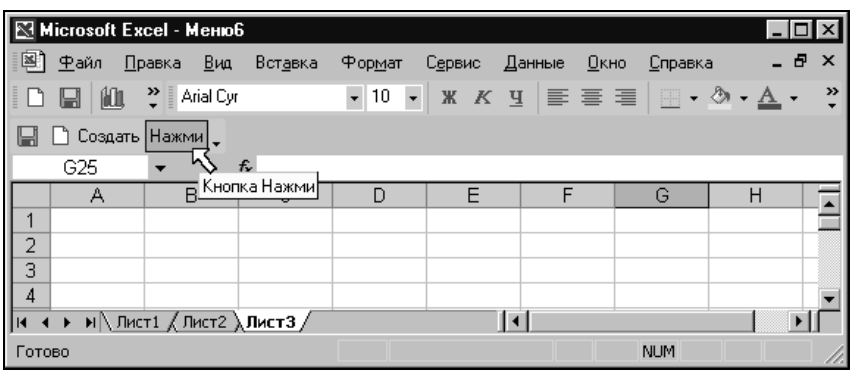


Рис. 9.5. Панель инструментов с тремя кнопками

Итак, в данном проекте (листинг 9.5):

- панель инструментов создается методом `Add` семейства `CommandBars` при условии, что значение параметра `MenuBar` установлено равным `False`;
- защита панели инструментов устанавливается свойством `Protection` объекта `CommandBar`. Если требуется, чтобы пользователь не мог ни перемещать, ни скрывать, ни видоизменять эту панель, то значение

свойства `Protection` должно быть положено равным `msoBarNoMove + msoBarNoChangeVisible + msoBarNoCustomize`;

- всем кнопкам назначены при помощи свойства `TooltipText` всплывающие подсказки;
- кнопки же имеют следующую структуру и функции:
  - кнопка **Сохранить**  позаимствована целиком (и внешний вид, и функции) из панели инструментов **Стандартная**;
  - для кнопки **Создать** позаимствована только картинка из одноименной кнопки панели инструментов **Стандартная**. Кроме картинки в этой кнопке выводится также надпись. С кнопкой связана процедура-заглушка, подтверждающая ее нажатие;
  - кнопка **Нажми**  не имеет картинки, на ней выводится только надпись. С ней также связана процедура-заглушка.

Листинг 9.5, *а*. Панель инструментов с кнопками. Модуль `ЭтаКнига`

```
Private Sub Workbook_Open()
   CommandBarBuilder
End Sub

Private Sub CommandBarBuilder()
   Dim cbMenuBar As CommandBar
   Dim btnSave As CommandBarButton
   Dim btnNew As CommandBarButton
   Dim btnPress As CommandBarButton
   Dim btnTmp As CommandBarButton
   Dim n As Integer

   Set cbMenuBar = Application.CommandBars.Add(Name:="МояПанель", _
                                               Position:=msoBarTop, _
                                               MenuBar:=False, _
                                               Temporary:=True)
   With cbMenuBar
      .Visible = True
      .Protection = msoBarNoMove + msoBarNoChangeVisible _
                  + msoBarNoCustomize
      With .Controls
         n = Application.CommandBars("Standard").Controls("Сохранить").ID
         Set btnSave = .Add(Type:=msoControlButton, _
                            ID:=n, _
                            Temporary:=True)
```

```
        Set btnNew = .Add(Type:=msoControlButton)
        Set btnPress = .Add(Type:=msoControlButton)
      End With
   End With

   btnSave.TooltipText = "Кнопка Сохранить"

   n = Application.CommandBars("Standard").Controls("Создать").ID
   Set btnTmp = _
             Application.CommandBars("Standard").FindControl( _
                                   Type:=msoControlButton, ID:=n)
   btnTmp.CopyFace
   With btnNew
      .TooltipText = "Кнопка Создать"
      .Style = msoButtonIconAndCaption
      .PasteFace
      .Caption = "Создать"
      .OnAction = "NewDoc"
   End With

   With btnPress
      .Caption = "Нажми"
      .TooltipText = "Кнопка Нажми"
      .Style = msoButtonCaption
      .OnAction = "PressMe"
   End With
End Sub

Private Sub Workbook_BeforeClose(Cancel As Boolean)
   Application.CommandBars("МояПанель").Delete
End Sub
```

Листинг 9.5, *б*. Панель инструментов с кнопками. Стандартный модуль

```
Public Sub NewDoc()
   MsgBox "New"
End Sub

Public Sub PressMe()
   MsgBox "Well done"
End Sub
```

Массив кнопок в панели инструментов

Использование не единичных кнопок, а целого массива может как упростить написание кода, так и сократить его длину. В следующем примере создается панель инструментов, состоящая из трех кнопок: **Первая**, **Вторая** и **Третья**. С этими кнопками связаны всплывающие подсказки `Это первая`, `Это вторая` и `Это третья`, и им ассоциированы процедуры `Press1`, `Press2`, `Press3`. В приводимом ниже коде (листинг 9.6) все перечисленные текстовые параметры кнопок (их имена, всплывающие подсказки и имена процедур) инкапсулированы в массиве `btnPar(2,2)` типа `String`, а сами кнопки — в массиве `btn(2)` типа `CommandBarButton`. Теперь всю работу с кнопками можно вести в операторе цикла.

Листинг 9.6, *а*. Массив кнопок в панели инструментов. Модуль `ЭтаКнига`

```
Private btnPar(2, 2) As String

Private Sub Workbook_Open()
   BtnParInit
   MenuBarBuilder
End Sub

Private Sub MenuBarBuilder()
   Dim cbMenuBar As CommandBar
   Dim btn(2) As CommandBarButton
   Set cbMenuBar = Application.CommandBars.Add(Name:="MyArrayBar", _
                                               Position:=msoBarTop, _
                                               MenuBar:=False, _
                                               Temporary:=True)
   Dim i As Integer
   For i = 0 To 2
      Set btn(i) = cbMenuBar.Controls.Add(Type:=msoControlButton)
      btn(i).Caption = btnPar(0, i)
      btn(i).TooltipText = btnPar(1, i)
      btn(i).Style = msoButtonCaption
      btn(i).OnAction = btnPar(2, i)
   Next
   cbMenuBar.Visible = True
End Sub

Private Sub BtnParInit()
   btnPar(0, 0) = "Первая"
   btnPar(0, 1) = "Вторая"
   btnPar(0, 2) = "Третья"
```

```
   btnPar(1, 0) = "Это первая"
   btnPar(1, 1) = "Это вторая"
   btnPar(1, 2) = "Это третья"
   btnPar(2, 0) = "Press1"
   btnPar(2, 1) = "Press2"
   btnPar(2, 2) = "Press3"
End Sub

Private Sub Workbook_BeforeClose(Cancel As Boolean)
   Application.CommandBars("MyArrayBar").Delete
End Sub
```

Листинг 9.6, *б*. Массив кнопок в панели инструментов. Стандартный модуль

```
Sub Press1()
   MsgBox "Press1"
End Sub

Sub Press2()
   MsgBox "Press2"
End Sub

Sub Press3()
   MsgBox "Press3"
End Sub
```

Раскрывающийся список панели инструментов и объект *CommandBarComboBox*

Объект `CommandBarComboBox` инкапсулирует в себе данные о раскрывающемся списке панели инструментов. В табл. 9.9 и 9.10 перечислены методы и свойства этого объекта, предоставляющие широкий спектр воздействия на список от его заполнения до очистки, а также различные возможности управления разнообразными параметрами списка от его размеров до стиля.

Таблица 9.9. *Методы объекта* `CommandBarComboBox`

Метод	Описание
`AddItem`	Добавляет элемент в список
`Clear`	Удаляет все элементы из списка

Таблица 9.9 *(окончание)*

Метод	Описание
`Copy`	Копирует список в панель инструментов
`Delete`	Удаляет список из семейства элементов управления
`Execute`	Выполняет процедуру, ассоциированную со списком
`Move`	Перемещает список в панель инструментов
`RemoveItem`	Удаляет элемент из списка
`Reset`	Переустанавливает встроенный список
`SetFocus`	Устанавливает фокус на списке

Таблица 9.10. *Свойства объекта* `CommandBarComboBox`

Свойство	Описание
`BuiltIn`	Проверяет, является ли список встроенным
`Caption`	Возвращает текст, отображаемый для поля, комбинированного со списком
`DescriptionText`	Возвращает описание элемента управления, используемое разработчиком в качестве пояснительного текста
`DropDownLines`	Число элементов одновременно без прокрутки отображаемых в списке
`DropDownWidth`	Ширина списка. Если значение свойства равно –1, то ширина основана на ширине самого длинного элемента списка. Если значение свойства равно 0, то на ширине элемента управления
`Enabled`	Определяет, является ли список достижимым для пользователя
`Height`, `Width`	Высота и ширина элемента управления
`Id`	Идентификатор списка
`Left`, `Top`	Координаты левого верхнего угла элемента управления
`List`	Возвращает элемент списка с указанным индексом
`ListHeaderCount`	Указывает число элементов списка, отображающихся выше разделительной линии
`ListIndex`	Индекс выбранного элемента
`OnAction`	Назначает списку макрос или процедуру

Таблица 9.10 *(окончание)*

Свойство	Описание
`Style`	Стиль списка. Допустимыми значениями могут быть следующие константы `MsoComboStyle`: `msoComboLabel` и `msoComboNormal`
`Tag`	Строка текста, предназначенная для служебного использования разработчиком
`Text`	Выбранный элемент списка
`TooltipText`	Всплывающая подсказка
`Visible`	Определяет, является ли список видимым

Панель инструментов с раскрывающимся списком

В этом разделе создается панель инструментов с единственным элементом управления — списком (рис. 9.6). В списке имеются три элемента: **Красный**, **Желтый** и **Зеленый**, выбор каждого из которых вызывает перекраску в соответствующий цвет ячеек рабочего листа. Кроме того, отображается диалоговое окно, подтверждающее сделанный выбор. Список в панель инструментов добавляется методом `Add` семейства `CommandBarControls` при значении параметра `Type` равным `msoControlDropdown`. Заполнение списка производится методом `AddItem`, а свойства `Text` и `ListIndex` возвращают выбранный элемент и его индекс (листинг 9.7).

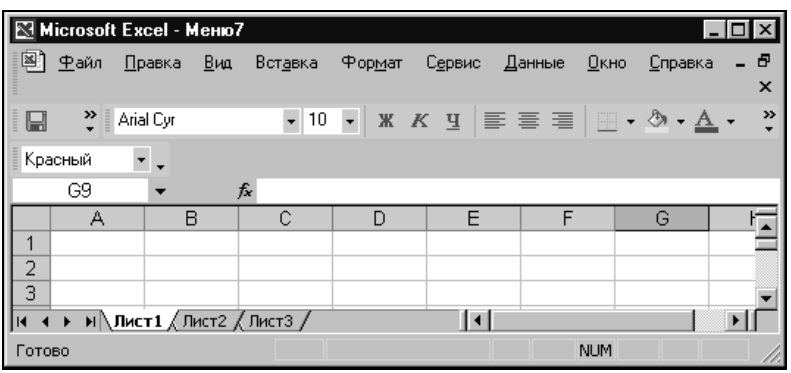


Рис. 9.6. Панель инструментов с раскрывающимся списком

Листинг 9.7, *а*. Панель инструментов с раскрывающимся списком. Модуль ЭтаКнига

```
Private Sub Workbook_Open()
   CommandBarBuilder
End Sub

Private Sub CommandBarBuilder()
   Dim cbMenuBar As CommandBar
   Set cbMenuBar = Application.CommandBars.Add(Name:="МояПанель", _
                                               Position:=msoBarTop, _
                                               MenuBar:=False, _
                                               Temporary:=True)
   Set cmbColor = cbMenuBar.Controls.Add(Type:=msoControlDropdown)
   With cmbColor
      .AddItem "Красный", 1
      .AddItem "Желтый", 2
      .AddItem "Зеленый", 3
      .ListIndex = 1
      .DropDownLines = 3
      .OnAction = "ChangeColor"
   End With
   cbMenuBar.Visible = True
End Sub

Private Sub Workbook_BeforeClose(Cancel As Boolean)
   Application.CommandBars("МояПанель").Delete
End Sub
```

Листинг 9.7, *б*. Панель инструментов с раскрывающимся списком. Стандартный модуль

```
Public cmbColor As CommandBarComboBox

Sub ChangeColor()
   Select Case cmbColor.ListIndex
      Case 1
         ActiveSheet.Cells.Interior.Color = RGB(255, 0, 0)
      Case 2
         ActiveSheet.Cells.Interior.Color = RGB(255, 255, 0)
      Case 3
         ActiveSheet.Cells.Interior.Color = RGB(0, 255, 0)
   End Select
   MsgBox cmbColor.Text
End Sub
```

Событие, связанное с раскрывающимся списком

С раскрывающимся списком связано событие `Change`, генерируемое при смене выбранного элемента, процедура обработки которого имеет следующий синтаксис:

```
Private Sub CommandBarComboBox_Change(ByVal Ctrl As CommandBarComboBox)
```

Управлять списком можно как свойством `OnAction`, так и посредством данной процедуры. В этом случае надо создать класс, инкапсулирующий в себе действия, связанные с выбором элементов, и назначить экземпляру данного класса ссылку на список. Например, меню со списком из предыдущего раздела можно было бы запрограммировать следующим образом. В классе `ComboBoxHandler` переменная `ComboBoxEvent`, которая будет давать ссылку на список, должна быть объявлена с помощью ключевого слова `WithEvents`, т. к. для этой переменной предусмотрена переменная обработки события `Change` (листинг 9.8)

Листинг 9.8, *а*. Панель инструментов с раскрывающимся списком. Модуль ЭтаКнига

```
Private cbHandler As New ComboBoxHandler

Private Sub Workbook_Open()
   Dim cbMenuBar As Office.CommandBar
   Dim cmbColor As Office.CommandBarComboBox
   Set cbMenuBar = Application.CommandBars.Add(Name:="МояПанель", _
                                               Position:=msoBarTop, _
                                               MenuBar:=False, _
                                               Temporary:=True)
   Set cmbColor = cbMenuBar.Controls.Add(Type:=msoControlDropdown)
   With cmbColor
      .AddItem "Красный", 1
      .AddItem "Желтый", 2
      .AddItem "Зеленый", 3
      .ListIndex = 1
      .DropDownLines = 3
   End With
   cbHandler.SyncBox cmbColor
   cbMenuBar.Visible = True
End Sub

Private Sub Workbook_BeforeClose(Cancel As Boolean)
   Application.CommandBars("МояПанель").Delete
End Sub
```

Листинг 9.8, *б*. Панель инструментов с раскрывающимся списком. Модуль класса `ComboBoxHandler`

```
Private WithEvents ComboBoxEvent As Office.CommandBarComboBox

Public Sub SyncBox(box As Office.CommandBarComboBox)
   Set ComboBoxEvent = box
End Sub

Private Sub Class_Terminate()
   Set ComboBoxEvent = Nothing
End Sub

Private Sub ComboBoxEvent_Change(ByVal Ctrl As Office.CommandBarComboBox)
   Select Case Ctrl.ListIndex
      Case 1
         ActiveSheet.Cells.Interior.Color = RGB(255, 0, 0)
      Case 2
         ActiveSheet.Cells.Interior.Color = RGB(255, 255, 0)
      Case 3
         ActiveSheet.Cells.Interior.Color = RGB(0, 255, 0)
   End Select
   MsgBox Ctrl.Text
End Sub
```

Конструирование контекстного меню

Методы создания строк меню и их элементов управления можно применить к разработке контекстного меню. Для построения контекстного меню надо:

1. Создать контекстное меню.
2. Добавить в контекстное меню элементы управления и связать с ним макросы.
3. Задать момент вывода контекстного меню.
4. Позаботится о том, чтобы после вывода пользовательского контекстного меню не отображалось встроенное в Excel контекстное меню.

Первые два пункта намеченной программы реализуются точно так же, как и при построении обычного меню. Созданное контекстное меню отображается на экране методом `ShowPopup` объекта `CommandBar`. Для задания момента отображения контекстного меню лучше всего подходит процедура обработки события `SheetBeforeRightClick` объекта `Workbook`. Параметры этой процедуры `Sh` и `Target` позволяют локализовать место в рабочей книге, в кото-

ром это меню должно отображаться, а именно рабочий лист и диапазон. Если необходимо отображение контекстного меню при нажатии правой кнопки мыши в произвольном месте любого рабочего листа книги, то о значениях параметров `Sh` и `Target` нет нужды беспокоиться. Установив значение параметра `Cancel` процедуры обработки события `SheetBeforeRightClick` объекта `Workbook` равным `True`, можно отключить выполнения тех функций, которые по умолчанию связаны с нажатием правой кнопки мыши.

Следующий код (листинг 9.9) демонстрирует технику создания контекстного меню на примере меню, состоящего из трех элементов: **Открыть**, **Сохранить** и **Создать**, причем эти элементы выполняют функции одноименных кнопок панели инструментов **Стандартная**. Отображение пользовательского контекстного меню произойдет при щелчке правой кнопкой в диапазоне **A1:B2**.

Листинг 9.9. Контекстное меню. Модуль `ЭтаКнига`

```
Private cbPopUp As CommandBar

Private Sub Workbook_Open()
   PopUpBarBuilder
End Sub

Private Sub PopUpBarBuilder()
   Dim n As Integer

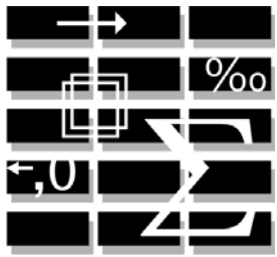
   Dim btnOpen As CommandBarButton
   Dim btnSave As CommandBarButton
   Dim btnNew As CommandBarButton

   Set cbPopUp = Application.CommandBars.Add(Name:="CustomBarPopup", _
                                             Position:=msoBarPopup)
   n = Application.CommandBars("Standard").Controls("Открыть").ID
   Set btnOpen = cbPopUp.Controls.Add(Type:=msoControlButton, _
                                      ID:=n, _
                                      Temporary:=True)
   n = Application.CommandBars("Standard").Controls("Сохранить").ID
   Set btnSave = cbPopUp.Controls.Add(Type:=msoControlButton, _
                                      ID:=n, _
                                      Temporary:=True)
   n = Application.CommandBars("Standard").Controls("Создать").ID
   Set btnNew = cbPopUp.Controls.Add(Type:=msoControlButton, _
                                      ID:=n, _
                                      Temporary:=True)
End Sub
```

```
Private Sub Workbook_BeforeClose(Cancel As Boolean)
   Application.CommandBars("CustomBarPopup").Delete
End Sub

Private Sub Workbook_SheetBeforeRightClick(ByVal Sh As Object, _
                                           ByVal Target As Range, _
                                           Cancel As Boolean)
   If Not (Application.Intersect(Target, Range("a1:b2")) Is Nothing) Then
      cbPopUp.ShowPopup
      Cancel = True
   End If
End Sub
```

ГЛАВА 10



Обработка данных: консолидация, сортировка, сценарии, фильтрация, сводные таблицы, поиск решения

MS Excel предоставляет в распоряжение разработчика весьма эффективные средства по обработке и анализу данных, а именно по их консолидации, сортировке и фильтрации, подведению промежуточных итогов, конструированию сценариев и структур, а также созданию сводных таблиц и диаграмм. Встроенный в MS Excel мощный инструмент — Поиск решения — позволяет с легкостью решать оптимизационные задачи линейного и нелинейного программирования. Применение же соответствующих объектных моделей обеспечивает создание полностью автоматизированных проектов как по обработке и анализу данных, так и по принятию решения.

Консолидация данных

При консолидации данных объединяются значения из нескольких диапазонов данных, происходит обобщение однородных данных. Например, можно обработать сведения, поступающие из различных отделов компании, и, таким образом, получить общую картину. Однако консолидация — это не только суммирование. В ходе этого процесса можно вычислить такие статистические величины, как среднее значение, стандартное отклонение, количество значений. Консолидировать данные в MS Excel можно несколькими способами:

- *консолидация при помощи трехмерных формул.* В этом случае создаются трехмерные формулы, содержащие ссылки на ячейки обобщаемых данных в разных диапазонах, возможно, находящихся на различных листах;
- *консолидация по расположению.* Ее следует использовать в случае, если данные всех исходных областей находятся в одном месте и размещены

в одинаковом порядке, например, если имеются данные из нескольких листов, созданных на основе одного шаблона;

- *консолидация по категориям*. Она применяется в случае, если требуется обобщить набор листов, имеющих одинаковые заголовки рядов и столбцов, но различную организацию данных. Этот способ позволяет консолидировать данные с одинаковыми заголовками со всех листов.

Консолидация при помощи трехмерных формул на рабочем листе

В качестве примера консолидации при помощи трехмерных формул рассмотрим бизнес-ситуацию построения консолидирующей таблицы о расходах фирмы ООО "Бескрайние просторы" за отчетный период с января по март, которые собраны в таблицы, расположенные на рабочих листах **Январь**, **Февраль** и **Март**. Расходы фирмы детализированы поквартально (рис. 10.1).

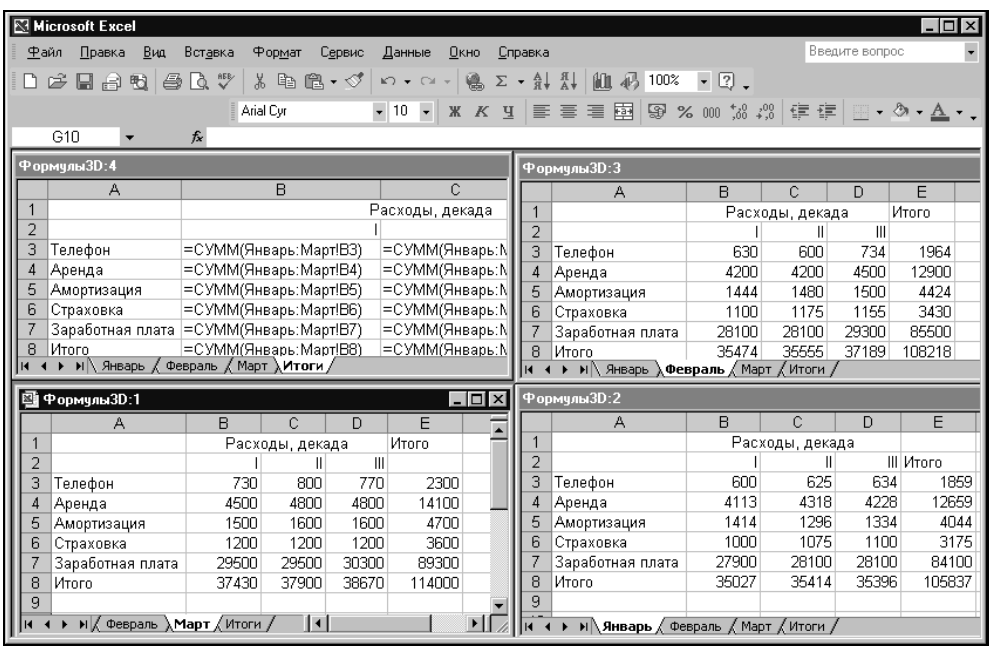


Рис. 10.1. Консолидация данных

Итак:

1. Создайте рабочий лист **Итоги**, на котором разместите шаблон отчетной таблицы.

2. Введите в ячейку **B3** формулу, находящую суммарные расходы за телефон в первом квартале с января по март:

   ```
   =СУММ(Январь:Март!B3)
   ```

   или равносильную формулу

   ```
   =СУММ(Январь!B3;Февраль!B3;Март!B3)
   ```

3. Расположите указатель мыши на маркере заполнения и переместите его вниз и вправо на диапазон **B3:E8**. Это позволит найти суммарные расходы по каждой категории расходов с июня по август.

Консолидация при помощи трехмерных формул в коде

Описанную в предыдущем разделе процедуру создания консолидирующей таблицы на основе трехмерных формул можно автоматизировать при помощи следующего кода (листинг 10.1). В нем имеется проверка наличия в книге листа с именем **Итоги**. Если такой лист отсутствует, на экране отображается сообщение, а процесс построения консолидирующей таблицы прерывается.

Листинг 10.1. Консолидация при помощи трехмерных формул

```
Sub DemoConsolidate3D()
    Dim rgn As Range
    Dim ws As Worksheet
    Dim str As String
    Dim nm As String
    nm = "Итоги"
    For Each ws In Worksheets
       str = str & ws.Name & "!B3" & ";"
       If ws.Name = nm Then
          MsgBox "Итоговый лист уже существует"
          Exit Sub
       End If
    Next
    str = Left(str, Len(str) - 1)

    Worksheets.Add After:=Worksheets(Worksheets.Count)
    ActiveSheet.Name = nm
    Worksheets(1).Range("A1:E8").Copy Worksheets(nm).Range("A1:E8")
    Range("B3:E8").Clear

    Range("B3").FormulaLocal = "=СУММ(" & str & ")"
    Range("B3").AutoFill Destination:=Range("B3:B8"), Type:=xlFillDefault
```

```
    Range("B3:B8").AutoFill Destination:=Range("B3:E8"), _
                            Type:=xlFillDefault
End Sub
```

Консолидация данных по положению и категориям

Консолидация данных по положению производится, если планируется объединение данных, находящихся в одинаковых ячейках разных диапазонов. Консолидация по категориям производится, если имеется нескольких диапазонов и планируется объединять эти данные по строкам или столбцам с одинаковыми подписями.

Совместно с консолидацией полезно использовать структурирование, причем структура может создаваться автоматически. Важно, что предназначенные для консолидации рабочие листы совсем не обязаны иметь одну и ту же структуру.

Опишем процесс консолидации данных на примере построения консолидирующей таблицы о расходах фирмы ООО "Бескрайние просторы" за отчетный период с января по март.

1. Убедитесь, что все диапазоны консолидируемых данных представлены в формате *списка*. Список — это набор строк листа, содержащий однородные данные, например набор адресов и телефонов клиентов. Первая строка в этом списке содержит заголовки столбцов. Первая строка каждого столбца содержит подпись, остальные строки — однотипные данные. Пустые строки или столбцы в списке отсутствуют. Если консолидация выполняется по положению, убедитесь, что макеты всех диапазонов совпадают. Если консолидация выполняется по категории, убедитесь, что подписи столбцов или строк, которые требуется объединить, совпадают (с учетом регистра букв).
2. Выберите левую верхнюю ячейку диапазона, в котором требуется разместить консолидированные данные. В нашем случае выберем ячейку **A2** рабочего листа **Итоги**.
3. Выберите команду **Данные | Консолидация**. На экране отобразится диалоговое окно **Консолидация** (рис. 10.2).
4. Выберите из раскрывающегося списка **Функция** так называемую *весовую*, или *итоговую функцию*. Эта функция задает тип вычисления, производимый при объединении данных в таблице консолидации. Допустимыми являются следующие типы: `Сумма`, `Количество`, `Среднее`, `Максимум`, `Минимум`, `Произведение`, `Количество чисел`, `Смещенное отклонение`, `Несмещенное отклонение`, `Смещенная дисперсия`, `Несмещенная дисперсия`. В данном случае выберите `Сумма`.

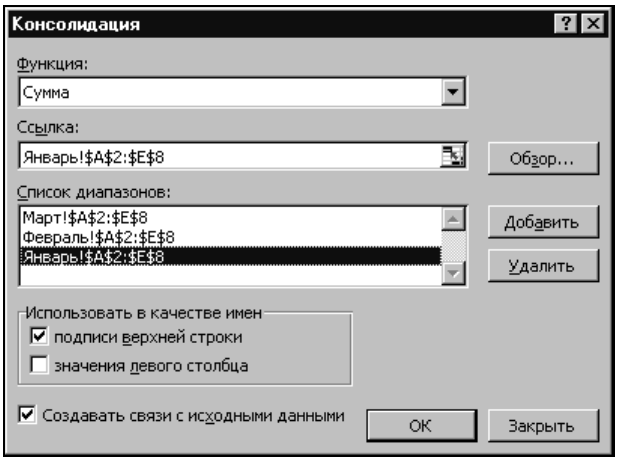


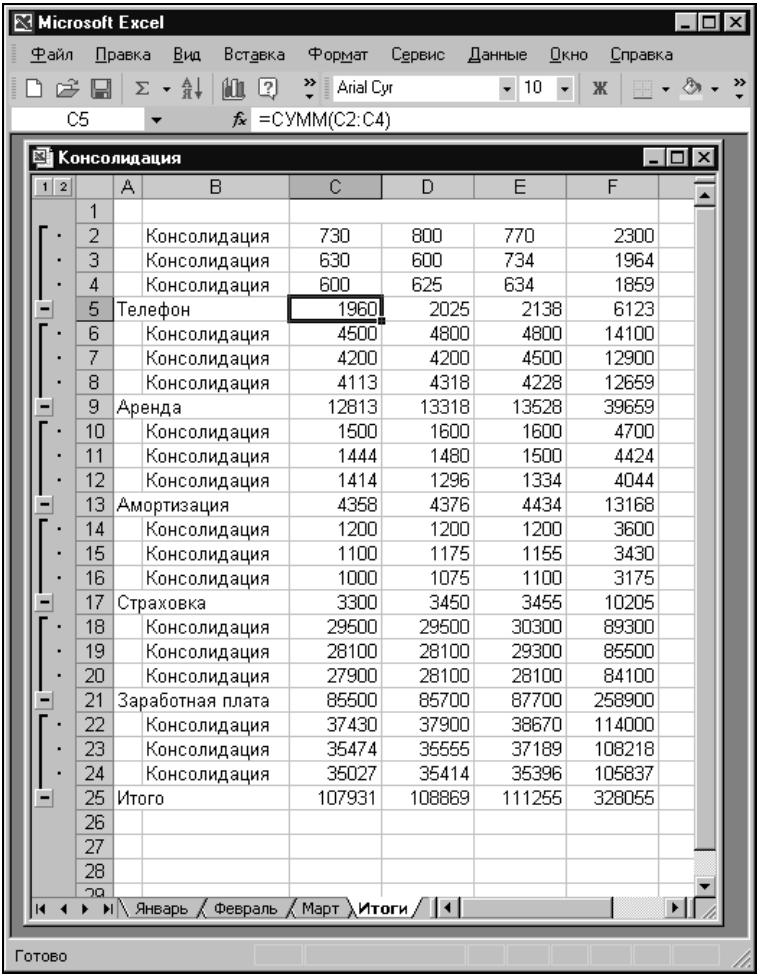
Рис. 10.2. Окно **Консолидация**

5. Щелкните поле **Ссылка**, откройте лист, содержащий первый диапазон данных для консолидации, введите ссылку на этот диапазон (в данном случае `Январь!$A$2:$E$8`) и нажмите кнопку **Добавить**. В результате ссылка на диапазон будет добавлена в список **Список диапазонов**. Повторите этот шаг для всех консолидируемых диапазонов. В данном случае для `Февраль!$A$2:$E$8` и `Март!$A$2:$E$8`. При этом:

   - если таблицу консолидации требуется обновлять автоматически при каждом изменении данных в каком-либо исходном диапазоне и позднее точно не потребуется изменять или добавлять диапазоны исходных данных для консолидации, установите флажок **Создавать связи с исходными данными**, что в нашем случае и следует сделать;
   - если консолидация выполняется по положению, оставьте все поля в группе **Использовать в качестве имен** пустыми. В MS Excel подписи исходных строк и столбцов не копируются в консолидированные данные. Если требуется скопировать подписи в консолидированные данные, сделайте это вручную. В нашем случае этот флажок не устанавливается;
   - если консолидация выполняется по категории, в группе **Использовать в качестве имен** установите флажки, соответствующие расположению подписей в исходных диапазонах: в верхней строке, в левом столбце или в верхней строке и в левом столбце одновременно. Все подписи, не совпадающие с подписями в других исходных областях, в консолидированных данных будут расположены в отдельных строках или столбцах. В нашем случае этот флажок устанавливается.

6. Нажмите кнопку **ОК**.

В результате будет создана консолидирующая таблица, показанная на рис. 10.3.



	A	B	C	D	E	F
1						
2		Консолидация	730	800	770	2300
3		Консолидация	630	600	734	1964
4		Консолидация	600	625	634	1859
5	Телефон		1960	2025	2138	6123
6		Консолидация	4500	4800	4800	14100
7		Консолидация	4200	4200	4500	12900
8		Консолидация	4113	4318	4228	12659
9	Аренда		12813	13318	13528	39659
10		Консолидация	1500	1600	1600	4700
11		Консолидация	1444	1480	1500	4424
12		Консолидация	1414	1296	1334	4044
13	Амортизация		4358	4376	4434	13168
14		Консолидация	1200	1200	1200	3600
15		Консолидация	1100	1175	1155	3430
16		Консолидация	1000	1075	1100	3175
17	Страховка		3300	3450	3455	10205
18		Консолидация	29500	29500	30300	89300
19		Консолидация	28100	28100	29300	85500
20		Консолидация	27900	28100	28100	84100
21	Заработная плата		85500	85700	87700	258900
22		Консолидация	37430	37900	38670	114000
23		Консолидация	35474	35555	37189	108218
24		Консолидация	35027	35414	35396	105837
25	Итого		107931	108869	111255	328055
26						
27						
28						

Рис. 10.3. Консолидирующая таблица

Методы и свойства, используемые при программировании консолидирующей таблицы

Для программного конструирования консолидирующей таблицы используется метод `Consolidate` объекта `Range`. Этот метод позволяет подвести итоги и обобщить однородные данные, размещенные в нескольких диапазонах. На рабочем листе действия, программируемые методом `Consolidate`, выполняются командой **Данные | Консолидация**.

```
expression.Consolidate(Sources, Function, TopRow, LeftColumn,
↳CreateLinks)
```

- *expression* — ссылка на диапазон или ячейку, расположенную в верхнем левом его углу, где будет построена консолидирующая таблица;
- *Sources* — необязательный параметр, задающий массив ссылок в формате **R1C1** на диапазоны, по которым строится консолидирующая таблица. Ссылки должны содержать полные имена диапазонов с указанием имен рабочих листов, на которых они расположены. Например, `Array("'Январь'!R1C1:R5C3", "'Февраль'!R1C1:R5C3");`
- *Function* — необязательный параметр, специфицирующий функцию, на основе которой строится консолидирующая таблица. Допустимыми значениями являются следующие константы `XlConsilidationFunction`: `xlAverage` (среднее), `xlCount` (количество значений), `xlCountNums` (количество чисел), `xlMax` (максимум), `xlMin` (минимум), `xlProduct` (произведение), `xlStDev` (несмещенная дисперсия), `xlStDevP` (смещенная дисперсия), `xlSum` (сумма), `xlVar` (несмещенное отклонение) и `xlVarP` (смещенное отклонение);
- *TopRow* — необязательный параметр, принимающий логические значения. Показывает, основывается ли консолидация на заголовках столбцов консолидируемых диапазонов;
- *LeftColumn* — необязательный параметр, принимающий логические значения. Показывает, основывается ли консолидация на заголовках строк консолидируемых диапазонов;
- *CreateLinks* — необязательный параметр, принимающий логические значения. Показывает, связана ли консолидируемая таблица с исходными таблицами. Если параметр принимает значения `True`, то консолидируемая таблица выводится в виде структуры.

При консолидации данных также важную роль играют три свойства объекта `Worksheet`, приведенные в табл. 10.1.

Таблица 10.1. *Свойства объекта* `Worksheet`*, используемые при консолидации данных*

Свойство	Описание
`ConsolidationOptions`	Возвращает трехмерный массив. Первый его элемент показывает, основывается ли консолидация на заголовках столбцов. Второй элемент устанавливает, основывается ли консолидация на заголовках строк консолидируемых диапазонов. Третий элемент показывает, связана ли консолидируемая таблица с исходными таблицами
`ConsolidationFunction`	Возвращает константу `XlConsolidationFunction`, идентифицирующую функцию, на основе которой строится консолидирующая таблица

Таблица 10.1 *(окончание)*

Свойство	Описание
ConsolidationSources	Возвращает массив ссылок на диапазоны, на основе которых была построена на рабочем листе консолидирующая таблица. Если на рабочем листе таковой таблицы нет, то это свойство возвращает значение Empty

Пример приложения, консолидирующего данные

Продемонстрируем на примере бизнес-ситуации с построением итоговой таблицы расходов фирмы ООО "Бескрайние просторы" за отчетный период, как в коде можно создавать и удалять консолидирующие таблицы. Для этого создайте рабочую книгу, имеющую несколько листов, скажем, **Январь**, **Февраль**, **Март**, с таблицами, как показано на рис. 10.1. Кроме того, в книге должен быть пустой лист **Итоги**. После этого в стандартный модуль и модуль ЭтаКнига надо добавить приводимый ниже код (листинг 10.2).

Листинг 10.2, *а*. Консолидация данных. Стандартный модуль

```
Sub ConsolidationBuilder()
   Dim nameSheet As String
   nameSheet = "Итоги"
   If WorksheetFunction.CountA(Worksheets(nameSheet).Columns(1)) > 0 Then
      MsgBox "Таблица на листе " & nameSheet & "уже существует"
      Exit Sub
   End If

   Dim rgn() As String
   ReDim rgn(1 To Worksheets.Count - 1)
   Dim ws As Worksheet
   Dim i As Integer
   i = 1
   For Each ws In Worksheets
      If ws.Name <> nameSheet Then
        rgn(i) = "'" & ws.Name & "'!R2C1:R8C5"
        i = i + 1
      End If
   Next

   Dim varRgn As Variant
   varRgn = rgn
```

```
   Worksheets(nameSheet).Range("A2").Select
   Selection.Consolidate Sources:=varRgn, _
                         Function:=xlSum, _
                         TopRow:=False, _
                         LeftColumn:=True, _
                         CreateLinks:=True
   Worksheets(nameSheet).Columns("A:F").AutoFit
End Sub

Sub ConsolidationKiller()
   Dim nameSheet As String
   nameSheet = "Итоги"
   If Not IsNull(Worksheets(nameSheet).Outline) Then
      Worksheets(nameSheet).Activate
      Range("A2").Select
      Selection.ClearOutline
      Cells.Select
      Selection.Clear
  End If
End Sub
```

Примечание

В программе производится консолидация заранее не оговоренного числа таблиц. Поэтому в качестве значения параметра `Sources` метода `Consolidate` нельзя приводить массив `Array`, размерность которого заранее не известна. Из данного затруднения в программе выходим очень просто — вводя дополнительную переменную типа `Variant`, ей присваиваем значения динамического массива с адресами консолидированных таблиц. А уж потом, в качестве значения параметра `Sources` используем значение этой вспомогательной переменной.

Листинг 10.2, *б*. Консолидация данных. Построение меню. Модуль `ЭтаКнига`

```
Private nameCB As String

Private Sub Workbook_Open()
   nameCB = "Консолидация"
   Dim cb As CommandBar
   Dim ccbConstructor As CommandBarButton
   Dim ccbKiller As CommandBarButton
   Set cb = Application.CommandBars.Add(Name:=nameCB, _
                                        Position:=msoBarTop, _
                                        MenuBar:=False, _
                                        Temporary:=True)
   With cb
      .Visible = True
```

```
      .Protection = msoBarNoMove + msoBarNoChangeVisible _
                    + msoBarNoCustomize
      With .Controls
         Set ccbConstructor = .Add(Type:=msoControlButton, _
                                   Temporary:=True)
         With ccbConstructor
            .Style = msoButtonCaption
            .Caption = "Консолидация"
            .OnAction = "ConsolidationBuilder"
         End With
         Set ccbKiller = .Add(Type:=msoControlButton, Temporary:=True)
         With ccbKiller
            .Style = msoButtonCaption
            .Caption = "Удаление консолидации"
            .OnAction = "ConsolidationKiller"
         End With
      End With
   End With
End Sub

Private Sub Workbook_BeforeClose(Cancel As Boolean)
   Application.CommandBars(nameCB).Delete
End Sub
```

В стандартном модуле имеются две процедуры, которые и реализуют бизнес-логику проекта:

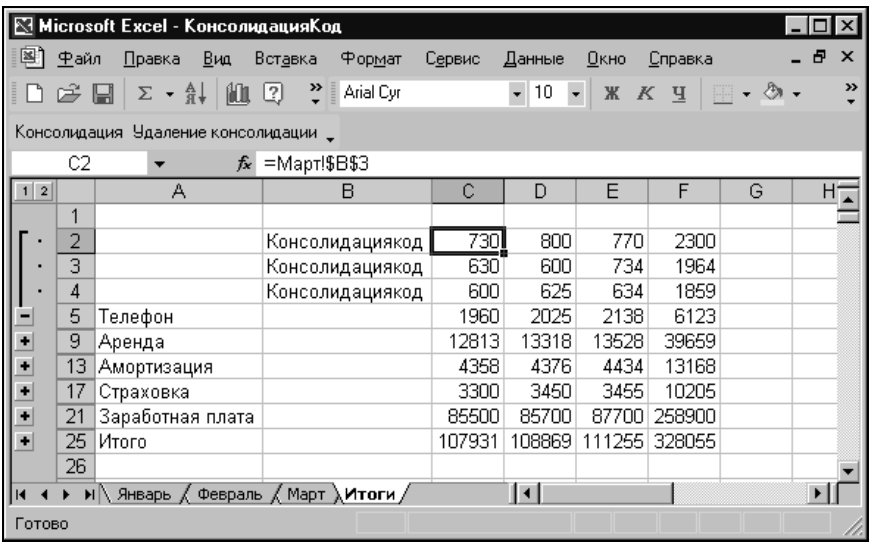
- процедура `ConsolidationBuilder` осуществляет построение консолидирующей таблицы по любому числу листов с данными, имена которых отличны от имени листа с консолидирующей таблицей, т. е. от имени **Итоги**. Прежде чем производить требуемые построения, эта процедура проверяет наличие на листе **Итоги** какой-либо таблицы (точнее, наличие каких-либо данных в его первом столбце). В случае присутствия таковых, построения не производятся;
- процедура `ConsolidationKiller` удаляет консолидирующую таблицу. Точнее, она методом `ClearOutline` удаляет структуру, созданную этой таблицей, а также методом `Clear` очищает содержимое ячеек. Прежде чем производить удаление, эта процедура проверяет наличие структуры на рабочем листе, и в случае отсутствия таковой, удаление отменяется за его ненадобностью.

В модуле `ЭтаКнига` имеются две процедуры, которые создают интерфейс проекта при открытии книги и его удаление при ее закрытии (рис. 10.4):

- процедура обработки события `Open` объекта `Workbook` конструирует панель инструментов **Консолидация**, на которой имеются две кнопки: **Консоли-**

дация и **Удаление консолидации**, которые вызывают выполнение процедур `ConsolidationBuilder` и `ConsolidationKiller`.

- процедура обработки события `BeforeClose` объекта `Workbook` удаляет панель **Консолидация** при закрытии книги.



	A	B	C	D	E	F	G	H
1								
2		Консолидациякод	730	800	770	2300		
3		Консолидациякод	630	600	734	1964		
4		Консолидациякод	600	625	634	1859		
5	Телефон		1960	2025	2138	6123		
9	Аренда		12813	13318	13528	39659		
13	Амортизация		4358	4376	4434	13168		
17	Страховка		3300	3450	3455	10205		
21	Заработная плата		85500	85700	87700	258900		
25	Итого		107931	108869	111255	328055		
26								

Рис. 10.4. Пример приложения, консолидирующего данные

Сортировка данных

Сортировка позволяет выстраивать данные в алфавитном или цифровом порядке по возрастанию или убыванию. MS Excel может сортировать строки списков и баз данных, а также столбцы рабочих листов. При сортировке текста в таблице можно сортировать как один столбец, так и таблицу целиком. Кроме того, можно выполнить сортировку по нескольким словам или полям в одном столбце таблицы. Например, если столбец содержит имена и фамилии, то можно сортировать его по имени или по фамилии, аналогично тому, как если бы имена и фамилии были в списке, а не в таблице.

Сортировка по выделенному полю

MS Excel позволяет сортировать список по выделенному полю. Для этого:

1. Выберите ячейку в сортируемом списке. MS Excel сам распознает список по выделенному полю.

2. Нажмите кнопку **Сортировать по возрастанию** или **Сортировать по убыванию**.

Сортировка выделенной области

Если MS Excel распознает при сортировке по выделенному полю не тот диапазон (фрагмент списка), который следует отсортировать, то первоначально надо выделить искомый диапазон, а затем произвести сортировку. При этом она будет осуществляться по первому столбцу из выделенного диапазона.

Сортировка по двум или трем столбцам

Рассмотрим процесс сортировки на примере данных таблицы **Сотрудники**, взятой из учебной базы данных **Борей**, поставляемой вместе с MS Office. Для того чтобы загрузить эту таблицу на рабочий лист, выберите команду **Файл | Открыть**. На экране отобразится окно **Открытие документа**. В списке **Тип файлов** выберите **Базы данных Access**. В списке папок и файлов выберите файл Борей.mdb, нажмите на кнопку **Открыть**. На экране отобразится окно **Выделить таблицу** со списком таблиц базы данных. Выберите таблицу **Сотрудники** и нажмите кнопку **OK**.

Итак, для сортировки по двум или трем столбцам:

1. Выделите ячейку внутри сортируемого списка или выделите его целиком. В нашем случае, например, выделите ячейку **A2**.
2. Выберите команду **Данные | Сортировка**. На экране отобразится диалоговое окно **Сортировка диапазона** (рис. 10.5).

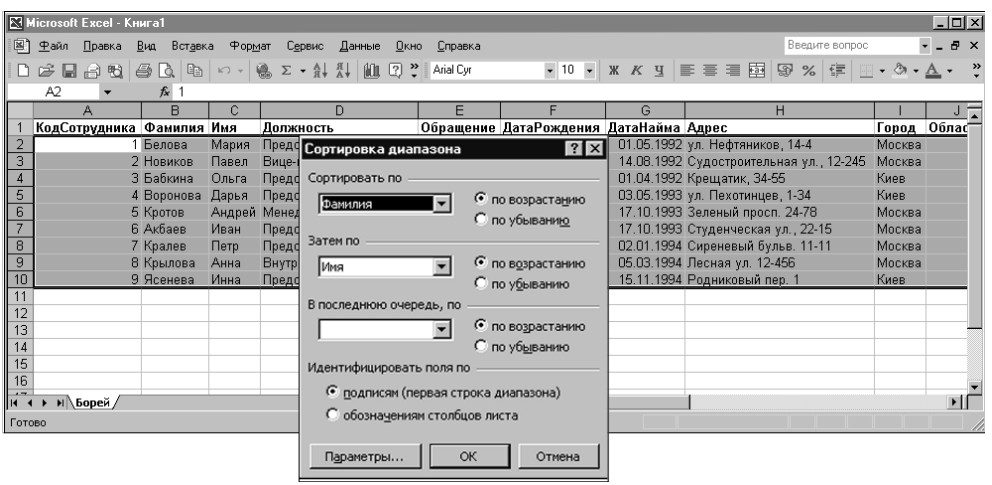


Рис. 10.5. Сортировка таблицы **Заказы** и диалоговое окно **Сортировка диапазона**

3. Выбор в списках **Сортировать по**, **Затем по**, **В последнюю очередь, по** определяет поля, используемые для упорядочения списка. Например, в телефонных книгах обычно записи сортируются сначала по фамилиям, потом по именам, а уж затем по отчествам. Всего в качестве ключей сортировки можно задать до трех полей. Переключатели **по возрастанию** и **по убыванию**, расположенные рядом с каждым из списков, определяют порядок сортировки. В нашем случае в списках выберите `Фамилия`, `Имя` и установите все переключатели в положение **по возрастанию**.
4. Нажмите кнопку **ОК**.

Сортировка по списку или по дням недели

В диалоговом окне **Параметры сортировки** (рис. 10.6), отображаемом на экране нажатием кнопки **Параметры** диалогового окна **Сортировка диапазона**, можно задать особый параметр сортировки, например по элементам списка, и, в частности, по дням недели, или потребовать учета регистра символа, или произвести сортировку не строк диапазона, а его столбцов.

Для сортировки по списку или по дням недели из списка **Сортировка по первому ключу** выберите тот список, по которому намерены произвести сортировку. Например, для сортировки по дням недели можно было бы выбрать список `Понедельник`, `Вторник`, `Среда`, `Четверг`, `Пятница`, `Суббота`, `Воскресенье`.

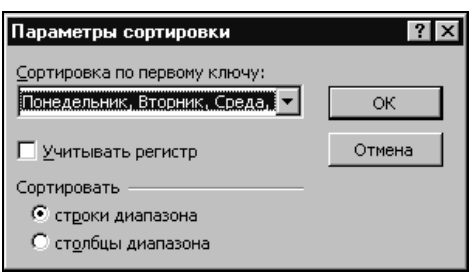


Рис. 10.6. Окно **Параметры сортировки**

Сортировка с учетом регистра

Для сортировки с учетом регистра в диалоговом окне **Параметры сортировки**, отображаемом на экране нажатием кнопки **Параметры** диалогового окна **Сортировка диапазона**, установите флажок **Учитывать регистр** и нажмите кнопку **ОК**. При сортировке по возрастанию с учетом регистра символы нижнего регистра предшествуют тем же символам верхнего.

Восстановление первоначального порядка вывода данных

Если в дальнейшем необходимо вернуться к первоначальному порядку сортировки данных, следует воспользоваться индексом. Конечно, если в списке данных нет индекса, то его предварительно надо ввести. Индекс — это специальное поле, содержащее уникальное значение для каждой записи. Например, порядковый номер, дату или время внесения записи. Когда понадобится вернуть первоначальный порядок данных, достаточно отсортировать их в порядке возрастания по полю индекса.

Сортировка на защищенном листе

Данные на защищенном листе можно сортировать, если они находятся в диапазоне, значение свойства `Locked` которого установлено равным `False`, и если параметр `AllowSorting` метода `Protect` установлен равным `True`. Свойство "только для чтения" `AllowSorting` объекта `Protection` возвращает значение этого параметра. Например, следующий код (листинг 10.3) разрешает сортировку диапазона **A1:A5** на защищенном листе, если она ранее не была установлена.

Листинг 10.3. Сортировка на защищенном листе

```
Sub DemoAllowSorting()
   ActiveSheet.Unprotect
   Range("A1:A5").Locked = False
   If ActiveSheet.Protection.AllowSorting Then
        ActiveSheet.Protect AllowSorting:=True
   End If
End Sub
```

Метод *Sort*

Метод `Sort` осуществляет сортировку данных с учетом до трех критериев, по которым производится сортировка. Метод `Sort` позволяет сортировать строки списков, сводных таблиц и баз данных, а также столбцы рабочих листов. Сортировка данных на рабочем листе производится пользователем командой **Данные | Сортировка**.

```
expression.Sort(Key1, Order1, Key2, Type, Order2, Key3, Order3,
↳Header, OrderCustom, MatchCase, Orientation, SortMethod, DataOption1,
↳DataOption2, DataOption3)
```

- *expression* — ссылка на ячейку диапазона или на сам диапазон, который будет сортироваться;

- *Key1* — необязательный параметр, задающий ссылку на первое упорядочиваемое поле;
- *Order1* — необязательный параметр, определяющий порядок упорядочивания поля, заданного параметром *Key1*. Допустимыми значениями являются следующие константы `XlSortOrder`: `xlAscending` (возрастающий порядок), `xlDescending` (убывающий);
- *Key2* — необязательный параметр, задающий ссылку на второе упорядочиваемое поле;
- *Type* — необязательный параметр, специфицирующий элементы, которые должны быть отсортированы. Используется только со сводными таблицами;
- *Order2* — необязательный параметр, определяющий порядок упорядочивания поля, заданного параметром *Key2*. Допустимыми значениями являются константы `XlSortOrder`;
- *Key3* — необязательный параметр, задающий ссылку на третье упорядочиваемое поле;
- *Order3* — необязательный параметр, определяющий порядок упорядочивания поля, заданного параметром *Key3*. Допустимыми значениями являются константы `XlSortOrder`;
- *Header* — необязательный параметр, специфицирующий, имеется ли в первой строке списка заголовок. Допустимыми значениями являются следующие константы `XlYesNoGuess`: `xlYes` (первая строка диапазона содержит заголовок, который не сортируется), `xlNo` (первая строка диапазона не содержит заголовок, по умолчанию считается данное значение), `xlGuess` (MS Excel решает сам, имеет ли список заголовок);
- *OrderCustom* — необязательный параметр, задающий пользовательский порядок сортировки. Является целым числом, указывающим порядковый номер списка, используемого в качестве шаблона сортировки;
- *MatchCase* — необязательный параметр, указывающий, надо ли учитывать регистры букв при сортировке;
- *Orientation* — необязательный параметр, специфицирующий ориентацию сортировки. Допустимыми значениями являются следующие константы `XlSortOrientation`: `xlTopToBottom` (сортировка осуществляется сверху вниз, т. е. по строкам), `xlLeftToRight` (сортировка осуществляется слева направо, т. е. по столбцам);
- *SortMethod* — необязательный параметр, указывающий метод сортировки. Применяется для таких языков, как китайский, японский;
- *DataOption1* — необязательный параметр, специфицирующий, как должен сортироваться текст в поле, заданном параметром *Key1*. Допустимы-

ми значениями являются следующие константы `XlSortDataOption`: `xlSortTextAsNumbers` (числовые и текстовые данные сортируются вместе), `xlSortNormal` (числовые и текстовые данные сортируются по раздельности);

- *`DataOption2`* — необязательный параметр, специфицирующий, как должен сортироваться текст в поле, заданном параметром *`Key2`*. Допустимыми значениями являются константы `XlSortDataOption`;
- *`DataOption3`* — необязательный параметр, специфицирующий, как должен сортироваться текст в поле, заданном параметром *`Key3`*. Допустимыми значениями являются константы `XlSortDataOption`.

Пример приложения, сортирующего данные

Рассмотрим пример простого приложения, в котором автоматизирован как процесс сортировки, так и процесс возвращения данных в исходный порядок. Сортировать будем данные из таблицы **Сотрудники** базы данных **Борей**. Для этого создайте рабочую книгу, состоящую из одного рабочего листа, в который импортируйте таблицу **Сотрудники**. После этого в стандартный модуль и модуль `ЭтаКнига` надо добавить код, приведенный в листинге 10.4).

Листинг 10.4, *а*. Сортировка данных из таблицы Сотрудники. Стандартный модуль

```
Public cbcb As CommandBarComboBox

Sub DoSort()
   Select Case cbcb.ListIndex
      Case 1
         Range("A2").Select
         Selection.Sort Key1:=Range("A1"), _
                        Order1:=xlAscending, Header:=xlYes
      Case 2
         Range("A2").Select
         Selection.Sort Key1:=Range("B2"), Order1:=xlAscending, _
                        Key2:=Range("C2"), Order2:=xlAscending, _
                        Key3:=Range("F2"), Order3:=xlAscending, _
                        Header:=xlYes
   End Select
   Range("A1").Select
End Sub
```

Листинг 10.4, *б*. Сортировка данных из таблицы Сотрудники. Модуль ЭтаКнига

```
Private Sub Workbook_Open()
   Dim cbSort As CommandBar
   Set cbSort = Application.CommandBars.Add(Name:="Сортировка", _
                            MenuBar:=True, Temporary:=True)
   With cbSort
      .Visible = True
      .Protection = msoBarNoMove + msoBarNoChangeVisible _
                  + msoBarNoCustomize
      With .Controls
      Set cbcb = .Add(Type:=msoControlDropdown)
         With cbcb
            .AddItem "Не отсортированные", 1
            .AddItem "Отсортированные", 2
            .ListIndex = 1
            .DropDownWidth = 120
            .OnAction = "DoSort"
         End With
      End With
   End With
End Sub

Private Sub Workbook_BeforeClose(Cancel As Boolean)
   Application.CommandBars("Сортировка").Delete
End Sub
```

Примечание

Обратите внимание, что после сортировки в коде выбирается одна ячейка. Это делается для того, чтобы снять выделение со всей таблицы.

В стандартном модуле имеется процедура и объявлена открытая переменная:

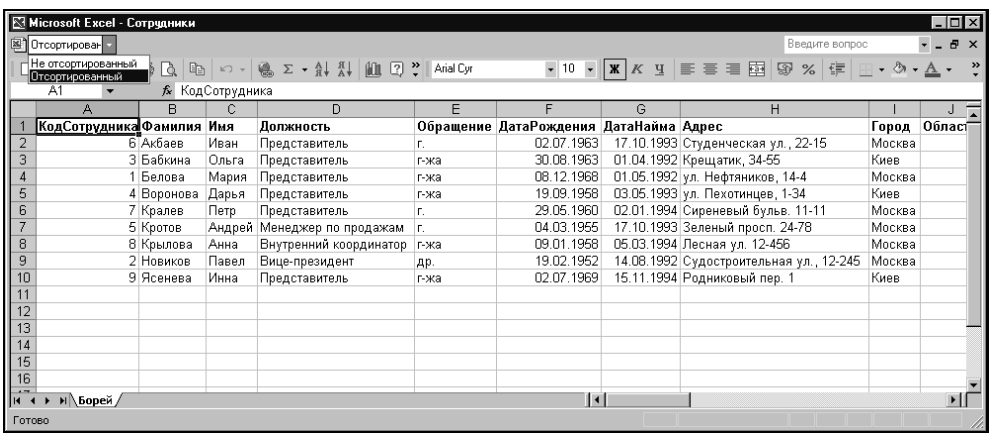
- процедура `ConsolidationBuilder` осуществляет построение консолидирующей таблицы по любому числу листов с данными, имена которых отличны от имени листа с консолидирующей таблицей, т. е. от имени **Итоги**. Прежде чем производить требуемые построения, эта процедура проверяет наличие на листе **Итоги** какой-либо таблицы (точнее, наличие каких-либо данных в его первом столбце). В случае присутствия таковых, построения не производятся;
- в открытой переменной `cbcb` объявлен объект `CommandBarComboBox`, который инкапсулирует в себе данные о раскрывающемся списке, управляю-

щем сортировкой из пользовательской панели инструментов. Процедура `DoSort` в зависимости от выбранного из раскрывающегося списка производит следующие действия:

- если выбран элемент `Не отсортированные` (значение индекса его в списке равно 1), то сортировка производится по полю **КодСотрудника**, тем самым восстанавливая первоначальную сортировку таблицы;
- если же выбран элемент `Отсортированные` (значение индекса его в списке равно 2), то сортировка производится по полям **Фамилия**, **Имя** и **ДатаРождения**.

В модуле `ЭтаКнига` имеются две процедуры, которые создают интерфейс проекта при открытии книги и его удаление при ее закрытии (рис. 10.7):

- процедура обработки события `Open` объекта `Workbook` конструирует панель инструментов **Сортировка**, на которой расположен раскрывающийся список с двумя элементами `Не отсортированные` и `Отсортированные`, выбор которых вызывает выполнение процедуры `ConsolidationBuilder` и `ConsolidationKiller`;
- процедура обработки события `BeforeClose` объекта `Workbook` удаляет панель **Консолидация** при закрытии книги.



	КодСотрудника	Фамилия	Имя	Должность	Обращение	ДатаРождения	ДатаНайма	Адрес	Город	Област
2	6	Акбаев	Иван	Представитель	г.	02.07.1963	17.10.1993	Студенческая ул., 22-15	Москва	
3	3	Бабкина	Ольга	Представитель	г-жа	30.08.1963	01.04.1992	Крещатик, 34-55	Киев	
4	1	Белова	Мария	Представитель	г-жа	08.12.1968	01.05.1992	ул. Нефтяников, 14-4	Москва	
5	4	Воронова	Дарья	Представитель	г-жа	19.09.1958	03.05.1993	ул. Пехотинцев, 1-34	Киев	
6	7	Кралев	Петр	Представитель	г.	29.05.1960	02.01.1994	Сиреневый бульв. 11-11	Москва	
7	5	Кротов	Андрей	Менеджер по продажам	г.	04.03.1955	17.10.1993	Зеленый просп. 24-78	Москва	
8	8	Крылова	Анна	Внутренний координатор	г-жа	09.01.1958	05.03.1994	Лесная ул. 12-456	Москва	
9	2	Новиков	Павел	Вице-президент	др.	19.02.1952	14.08.1992	Судостроительная ул., 12-245	Москва	
10	9	Ясенева	Инна	Представитель	г-жа	02.07.1969	15.11.1994	Родниковый пер. 1	Киев	

Рис. 10.7. Сортировка таблицы **Сотрудники**

Фильтрация

Фильтрация — это эффективный способ поиска подмножества данных и работы с ними в списке. В отфильтрованном списке отображаются только строки, отвечающие условиям отбора, заданным для столбца.

MS Excel предоставляет две команды для фильтрации списков:

- *автофильтр*, включая фильтр по выделенному, для простых условий отбора;
- *расширенный фильтр* для более сложных условий отбора.

При фильтрации порядок записей в списке не изменяется, строки, которые не удовлетворяют критерию фильтрации, временно скрываются. Отфильтрованные строки можно редактировать, форматировать и выводить на печать, а также создавать на их основе диаграммы, не перемещая их и не изменяя порядок строк.

Использование автофильтра

При использовании команды **Автофильтр** справа от подписей столбцов в фильтруемом списке появляются стрелки автофильтра. У отфильтрованных элементов стрелки автофильтра окрашиваются в синий цвет, а у неотфильтрованных они остаются черного цвета.

В раскрывающемся списке автофильтров можно выбрать (рис. 10.8):

- команду **Все** для отображения всех элементов списка;
- команду **Первые 10**, позволяющую отобразить некоторое количество (по умолчанию 10) наибольших (по умолчанию) или наименьших элементов списка;
- команду **Условие**, чтобы вызвать диалоговое окно **Пользовательский автофильтр**, где можно задать простой критерий, содержащий до двух условий;
- значение поля для поиска точного соответствия.



Рис. 10.8. Раскрывающийся список автофильтра

Описание процесса автофильтрации

Рассмотрим процесс автофильтрации на примере данных таблицы **Сотрудники**, взятой из учебной базы данных **Борей**.

1. Перед применением автофильтра убедитесь, что он не используется для какого-либо другого списка. Для этого выберите команду **Данные**

Фильтр. Если команда **Автофильтр** отмечена флажком, выберите ее, чтобы отменить автофильтр для другого списка.

2. Выделите ячейку внутри фильтруемого списка или выделите его целиком. В нашем случае, например, выделите ячейку **A2**.
3. Выберите команду **Данные | Фильтр | Автофильтр**. В результате справа от текста, отображаемого в ячейках с названиями полей в фильтруемом списке, появятся стрелки автофильтра, а сами эти ячейки превратятся в раскрывающиеся списки.
4. Раскройте список, соответствующий полю, которое следует включить в критерий. Выберите один из допустимых критериев:
   - **Все** (вывод на экран всех записей);
   - **Первые 10** (вывод на экран заданного числа или заданный процент первых или последних записей);
   - **Условие** (вывод на экран записей по условию, заданному в отображаемом диалоговом окне **Пользовательский автофильтр**);
   - **Точное значение** (вывод на экран записей, поля которых в точности совпадают с выбранным значением).

   Например, в нашем случае в списке **Должность** выберите точное значение **Представитель**.

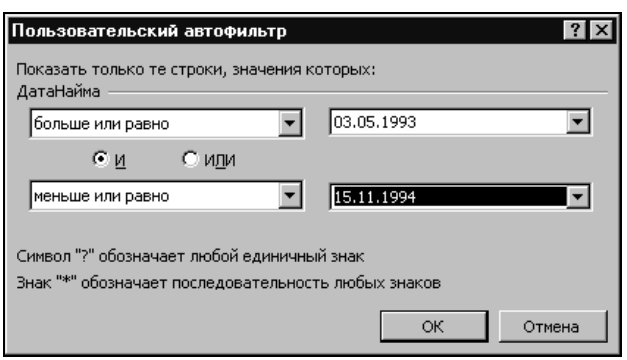


Рис. 10.9. Диалоговое окно **Пользовательский автофильтр**

5. Вернитесь к предыдущему шагу, если в критерий необходимо включить другое поле. Например, в списке **ДатаНайма** выберите **Условие**. На экране отобразится диалоговое окно **Пользовательский автофильтр** (рис. 10.9). Диалоговое окно **Пользовательский автофильтр** позволяет быстро задать более сложное условие, чем простое сравнение. В левом верхнем раскрывающемся списке выбирается операция сравнения (в данном случае выберите **больше или равно**), в правом — выбирается или вводится значение (введите `03.05.1993`). Затем, если необходимо, выберите один из пере-

ключателей **И**, **ИЛИ** и задайте вторую операцию сравнения и значение. В данном случае выберите переключатель **И**, операцию сравнения **меньше или равно** и введите значение `15.11.1994`.

6. Нажмите кнопку **ОК**. Результат автофильтрации будет немедленно отображен на экране (рис. 10.10).

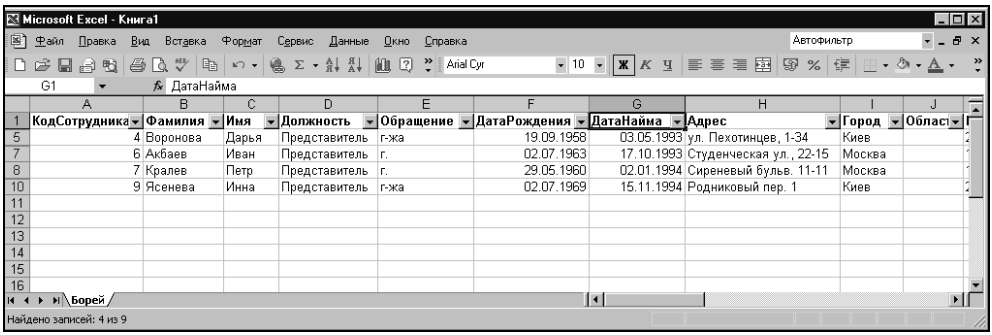


Рис. 10.10. Результат автофильтрации

Метод *AutoFilter*

Метод `AutoFilter` объекта `Range` программирует выполнение команды **Данные | Фильтр | Автофильтр** для фильтрации списка по критериям, указанным в параметрах.

```
expression.AutoFilter(Field, Criteria1, Operator,
↳Criteria2, VisibleDropDown)
```

- *expression* — ссылка на ячейку диапазона или на сам диапазон, который будет фильтроваться;
- *Field* — необязательный параметр, задающий номер поля списка, в котором производится фильтрация данных. Нумерация производится с крайнего левого поля, причем первое поле имеет номер 1;
- *Criteria1* и *Criteria2* — необязательные параметры, специфицирующие два возможных критерия фильтрации поля. Допускается использование строковой константы, например "`101`", и знаков отношений ">, <,>=, <=, =, <>". Используйте "=" для фильтрации пустых полей, а "<>" для фильтрации непустых полей. Если параметр опущен, то критерием является значение `All`. Если значение параметра *Operator* равно `xlTop10Items`, то параметр `Criteria1` определяет число отображаемых элементов (например, "`30`");
- *Operator* — необязательный параметр, допустимыми значениями которого могут быть следующие константы `XlAutoFilterOperator`: `xlAnd` (для ло-

гического объединения первого и второго критериев фильтрации), xlOr (для логического сложения первого и второго критериев), xlTop10Items (для отображения первых десяти элементов поля), xlBottom10Items (для отображения последних десяти элементов поля), xlTop10Percent (для отображения первых 10% элементов поля), xlBottom10Percent (для отображения последних 10% элементов поля);

- *VisibleDropDown* — необязательный параметр, принимающий логические значения. Определяет, отображаются ли раскрывающиеся списки. По умолчанию параметр принимает значение True.

Отображение и удаление кнопок со стрелками автофильтра

Метод AutoFilter объекта Range, примененный без параметров, приводит к отображению или скрытию кнопок со стрелками автофильтра.

Отображение всех отфильтрованных данных

При работе с фильтрами полезен метод ShowAllData объекта Worksheet, который отображает все отфильтрованные и не отфильтрованные строки рабочего листа. Например,

```
Worksheets("Заказы").ShowAllData
```

Как узнать, что на листе имеются кнопки со стрелками автофильтра

Свойство "только для чтения" AutoFilterMode объекта Worksheet принимает логические значения. Оно возвращает значение True, если на рабочем листе имеются кнопки со стрелками автофильтра.

Как узнать, что на листе имеются отфильтрованные данные

Свойство "только для чтения" FilterMode объекта Worksheet принимает логические значения. Оно возвращает значение True, если на рабочем листе имеются отфильтрованные данные со скрытыми строками.

Объекты *AutoFilter* и *Filter* и семейство *Filters*

Объект AutoFilter инкапсулирует в себе данные об используемом на рабочем листе автофильтре. Этот объект возвращается свойством AutoFilter объекта Worksheet. Основные свойства объекта AutoFilter приведены в табл. 10.2, а свойства объекта Filter — в табл. 10.3.

Таблица 10.2. Основные свойства объекта `AutoFilter`

Свойство	Описание
`Filters`	Возвращает семейство `Filters` объектов `Filter`, т. е. всех фильтров, образующих данный автофильтр. Свойства объекта `Filter` приведены в табл. 10.3. У семейства `Filters` основными свойствами являются `Count` и `Item`, которые возвращают число элементов и конкретный элемент семейства
`Range`	Возвращает диапазон, к которому применен фильтр

Таблица 10.3. Свойства объекта `Filter`

Свойство	Описание
`On`	Возвращает значение `True`, если фильтр установлен
`Criteria1`	Возвращает первый критерий фильтра
`Criteria2`	Возвращает второй критерий фильтра
`Operator`	Возвращает оператор фильтра. Допустимыми значениями являются константы `XlAutoFilterOperator`

В приводимом ниже коде (листинг 10.5) сначала определяется, имеется ли на рабочем листе автофильтр, а затем, в случае его наличия, находится как количество фильтров, его образующих, так и специфицированные критерии для установленных фильтров.

Листинг 10.5. Определение критериев для установленных фильтров

```
Sub DemoFilter ()
   Dim f As Filter
   Dim w As Worksheet
   Set w = Worksheets(1)
   Dim n As Integer ' Номер фильтра
   n = 1
   If w.AutoFilterMode Then
      MsgBox "Число фильтров: " & Worksheets(1).AutoFilter.Filters.Count
      For Each f In w.AutoFilter.Filters
         If f.On Then
            If f.Operator Then
               MsgBox n & vbTab & f.Criteria1 & vbTab & f.Criteria2
            Else
               MsgBox n & vbTab & f.Criteria1
            End If
         End If
```

```
      n = n + 1
      Next
   Else
      MsgBox "На рабочем листе нет автофильтра"
   End If
End Sub
```

Доступ к раскрывающимся спискам автофильтра на защищенном листе

Свойство `EnableAutoFilter` рабочего листа позволяет управлять доступом к раскрывающимся спискам автофильтра на защищенном листе. Если его значение равно `True`, то раскрывающиеся списки автофильтра достижимы для пользователя даже в случае установки защиты на рабочем листе. При этом должен быть включен режим защиты только пользовательского интерфейса:

```
ActiveSheet.EnableAutoFilter = True
ActiveSheet.Protect Contents:=True, UserInterfaceOnly:=True
```

Пример приложения, фильтрующего данные

Продемонстрируем использование метода `AutoFilter` на примере бизнес-ситуации создания приложения, фильтрующего по должностям таблицу **Сотрудники** учебной базой данных **Борей**. Итак, создайте книгу, состоящую из одного рабочего листа **Борей**, в который импортируйте таблицу **Сотрудники** базы данных **Борей**. При щелчке правой кнопкой на заголовке таблицы будет отображаться контекстное меню с единственной командой **Filter**. Выбор этой команды приведет к отображению окна со списком, в котором перечислены должности сотрудников, и флажком **Фильтрация**. При установленном флажке на рабочем листе фильтруются сотрудники по выбранной в списке должности. При снятом флажке — фильтрация удаляется (рис. 10.11).

Итак, создайте форму, на ней расположите список и флажок. Используя окно **Properties**, установите значение свойства `Name` формы равным `frmFilter`. В модулях формы, `ЭтаКнига` и стандартном, наберите приводимый ниже код (листинг 10.6).

Листинг 10.6, *а*. Фильтрация данных по одному выбранному сотруднику. Модуль формы

```
Private Sub UserForm_Initialize()
   CheckBox1.Caption = "Фильтрация"
   If ActiveSheet.AutoFilterMode Then
      Rows(1).Select
```

```
      Selection.AutoFilter
   End If
   Range("A2").Sort key1:=Range("D2"), header:=xlYes
   Dim rgn As Range
   Set rgn = Range("A1").CurrentRegion.Columns(4)
   Me.Caption = rgn.Cells(1).Value
   Set rgn = rgn.Offset(1, 0).Resize(rgn.Rows.Count – 1, 1)
   Dim c As Range
   Dim oldValue As String
   Dim curValue As String
   curValue = rgn.Cells(1).Value
   oldValue = curValue
   ListBox1.AddItem curValue
   For Each c In rgn.Cells
      curValue = c.Value
      If curValue <> oldValue Then
         ListBox1.AddItem curValue
         oldValue = curValue
      End If
   Next
   Range("A2").Sort key1:=Range("A2"), order1:=xlAscending, header:=xlYes
End Sub

Private Sub CheckBox1_Change()
   If CheckBox1.Value Then
      ListBox1_Change
   Else
      If ActiveSheet.AutoFilterMode Then
         Rows(1).Select
         Selection.AutoFilter
      End If
   End If
End Sub

Private Sub ListBox1_Change()
  If CheckBox1.Value Then
     If ActiveSheet.AutoFilterMode Then
        Rows(1).Select
        Selection.AutoFilter Field:=4, Criteria1:=ListBox1.Text
     Else
        Rows(1).Select
        Selection.AutoFilter
        Selection.AutoFilter Field:=4, Criteria1:=ListBox1.Text
      End If
  End If
End Sub
```

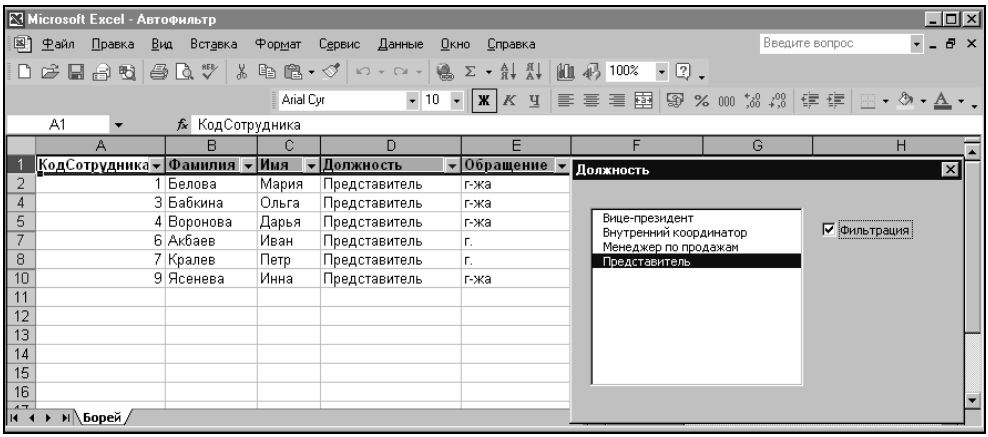


Рис. 10.11. Таблица **Сотрудники** и окно **Должность**

Листинг 10.6, *б*. Фильтрация данных по одному выбранному сотруднику. Модуль ЭтаКнига

```
Private cb As CommandBar

Private Sub Workbook_Open()
   Dim cbb As CommandBarButton
   Set cb = Application.CommandBars.Add(Name:="Filter", _
                                        Position:=msoBarPopup)
   Set cbb = cb.Controls.Add(Type:=msoControlButton)
   With cbb
      .Style = msoButtonCaption
      .Caption = "Filter"
      .OnAction = "DoFilter"
   End With
End Sub

Private Sub Workbook_BeforeClose(Cancel As Boolean)
   Application.CommandBars("Filter").Delete
End Sub

Private Sub Workbook_SheetBeforeRightClick(ByVal Sh As Object, _
      ByVal Target As Range, Cancel As Boolean)
   Cancel = True
   If Sh.Name = "Борей" Then
      If Not (Application.Intersect(Target, Rows(1)) Is Nothing) Then
```

```
            cb.ShowPopup
         End If
      End If
End Sub
```

Листинг 10.6, *в*. Фильтрация данных по одному выбранному сотруднику. Стандартный модуль

```
Sub DoFilter()
   frmFilter.Show
End Sub
```

В модуле формы имеются три процедуры:

- процедура обработки события `Initialize` формы устанавливает значения свойства `Caption` флажка и формы, а также заполняет список. Список заполняется из поля **Должность** таблицы, причем в процессе заполнения повторяющиеся данные из этого поля опускаются;
- процедура обработки события `Change` списка при выборе элемента из списка, если флажок **Фильтрация** установлен, фильтрует таблицу, причем, в случае если кнопки автофильтра еще не были выведены в таблицу, то эта процедура сначала инициализирует автофильтр, а затем производит фильтрацию;
- процедура обработки события `Change` флажка в зависимости от состояния флажка либо производит фильтрацию, вызывая процедуру обработки события `Change` списка, либо удаляет автофильтр.

В модуле `ЭтаКнига` имеются три процедуры:

- процедура обработки события `Open` объекта `Workbook` создает контекстное меню, в котором имеется единственный элемент **Filter** и с которым связывается процедура `DoFilter`;
- процедура обработки события `SheetBeforeRightClick` объекта `Workbook` отображает контекстное меню при щелчке пользователем правой кнопкой на первой строке листа **Борей**;
- процедура обработки события `BeforeClose` объекта `Workbook` удаляет из книги контекстное меню при закрытии книги.

В стандартном модуле имеется единственная процедура `DoFilter`, которая отображает окно **Должность**.

Промежуточные итоги

В MS Excel имеется средство, позволяющее получить предварительные результаты, если требуется объединить данные в отдельные группы, например, сгруппировать продажи по номеру счета или проекты по именам руководи-

телей. При вставке автоматических промежуточных итогов MS Excel изменяет разметку списка, что позволяет отображать и скрывать строки каждого промежуточного итога.

Промежуточные итоги позволяют обобщить данные. Предположим, имеется список продаж. Он включает в себя дату продажи, номер счета, товар, количество проданных единиц товара, цену. Использование команды **Итоги** позволит, не составляя формул и не преобразуя таблицу, по предлагаемому MS Excel списку математических выражений найти промежуточные и общие итоги, структурировать данные и вставить в таблицу строки с промежуточными и общими итогами.

Перед тем как вставить промежуточные итоги, необходимо отсортировать список, чтобы сгруппировать строки, по которым нужно подвести итоги. После этого можно подсчитать промежуточные итоги любого столбца, содержащего числа.

Описание процесса создания промежуточных итогов

Рассмотрим процесс создания промежуточных итогов на примере данных таблицы **Заказы** учебной базы данных **Борей**, точнее, полей **КодЗаказа**, **СтоимостьДоставки**, **НазваниеПолучателя**, **ГородПолучателя**, **СтранаПолучателя**.

1. Прежде чем подводить промежуточные итоги, убедитесь, что записи отсортированы по тем параметрам, по которым итоги будут подводиться. В данном случае по полю **СтранаПолучателя**.
2. Выделите ячейку списка, а затем выберите команду **Данные | Итоги**. На экране отобразится окно **Промежуточные итоги** (рис. 10.12).
3. Укажите, как группировать данные, выбрав значение в списке **При каждом изменении в**. Список содержит названия столбцов базы данных или списка. Например, мы будем определять суммарную стоимость доставки заказов в каждую из стран, поэтому выберите **СтранаПолучателя**.
4. Выберите операцию, выполняемую над данными, из списка **Операция**. Возможно подведение итогов по одной из следующих операций: `Сумма`, `Количество`, `Среднее`, `Максимум`, `Минимум`, `Произведение`, `Количество чисел`, `Смещенное отклонение`, `Несмещенное отклонение`, `Смещенная дисперсия`, `Несмещенная дисперсия`. В нашем случае выберите `Сумма`.
5. Выберите данные, которые должны участвовать в расчетах, установив в списке **Добавить итоги по** флажки требуемых столбцов. В нашем случае установите флажок **СтоимостьДоставки**. Если требуется подвести итоги по нескольким столбцам одновременно, установите флажки для каждого из них.

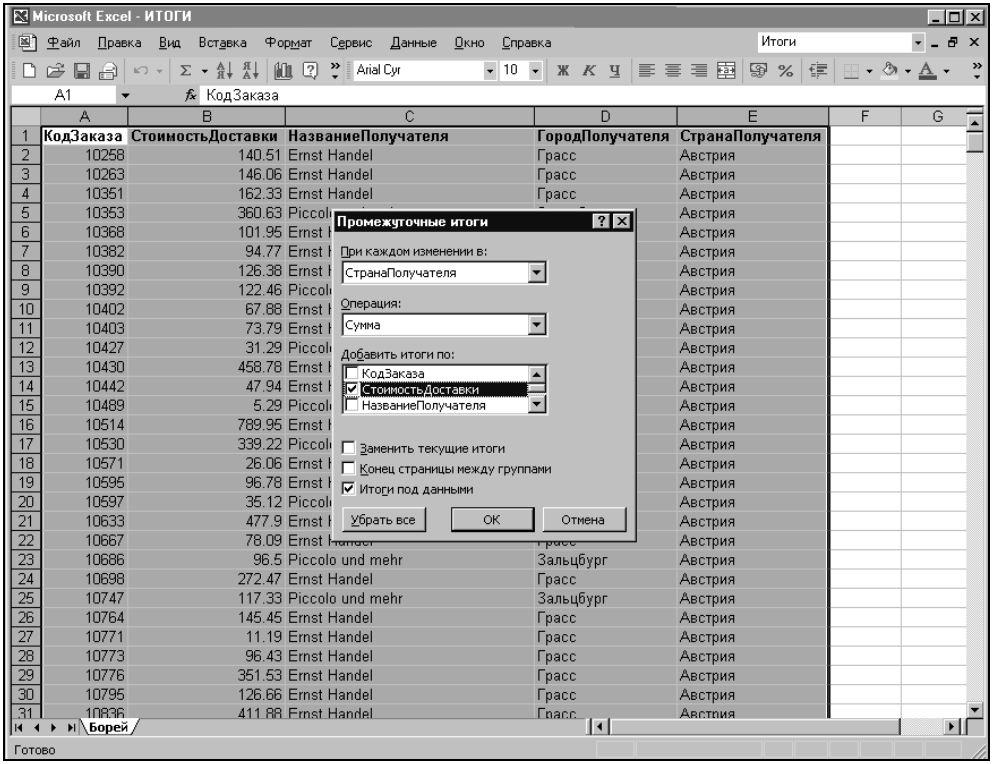


Рис. 10.12. Окно **Промежуточные итоги**

6. Чтобы заменить все старые промежуточные итоги на вновь созданные, устанавливается флажок **Заменить текущие итоги**. В нашем случае этот флажок должен быть снят.

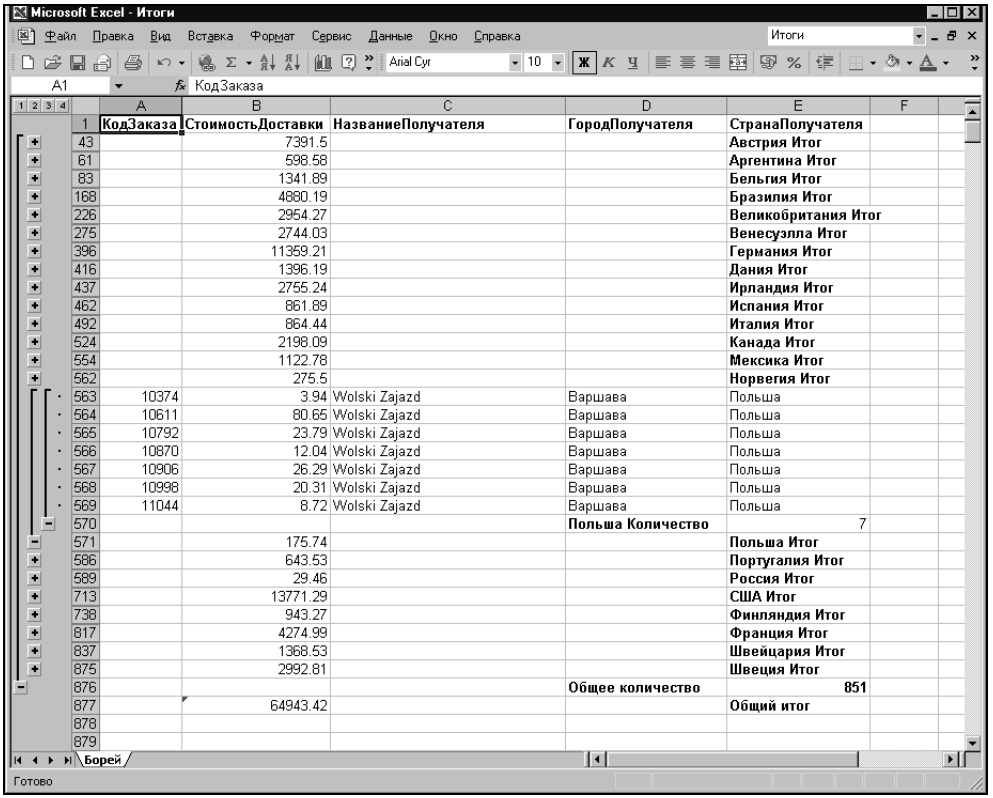
7. Чтобы вставить символ конца страницы после каждой группы, для которой подводятся итоги, устанавливается флажок **Конец страницы между группами**. В данном случае этот флажок должен быть снят.

8. По умолчанию строки, содержащие общие и промежуточные итоги, располагаются под данными. Если флажок **Итоги под данными** сбросить, то итоги будут располагаться над данными.

9. Нажмите кнопку **OK**.

10. Если необходимо к полученным итогам добавить дополнительные *(вложенные итоги)*, то надо повторить шаги со 2 по 9, при этом флажок **Заменить текущие итоги** должен быть сброшен. Например, добавим в промежуточные итоги информацию о суммарном количестве заказов по каждой из стран. С этой целью в окне **Промежуточные итоги** в списке **При каждом изменении в** выберите **СтранаПолучателя**, в списке **Опера-**

ция выберите `Количество`, а в списке **Добавить итоги по** установите флажок **СтранаПолучателя**. Нажмите кнопку **ОК** (рис. 10.13).



	A	B	C	D	E	F
1	КодЗаказа	СтоимостьДоставки	НазваниеПолучателя	ГородПолучателя	СтранаПолучателя	
43		7391.5			Австрия Итог	
61		598.58			Аргентина Итог	
83		1341.89			Бельгия Итог	
168		4880.19			Бразилия Итог	
226		2954.27			Великобритания Итог	
275		2744.03			Венесуэлла Итог	
396		11359.21			Германия Итог	
416		1396.19			Дания Итог	
437		2755.24			Ирландия Итог	
462		861.89			Испания Итог	
492		864.44			Италия Итог	
524		2198.09			Канада Итог	
554		1122.78			Мексика Итог	
562		275.5			Норвегия Итог	
563	10374	3.94	Wolski Zajazd	Варшава	Польша	
564	10611	80.65	Wolski Zajazd	Варшава	Польша	
565	10792	23.79	Wolski Zajazd	Варшава	Польша	
566	10870	12.04	Wolski Zajazd	Варшава	Польша	
567	10906	26.29	Wolski Zajazd	Варшава	Польша	
568	10998	20.31	Wolski Zajazd	Варшава	Польша	
569	11044	8.72	Wolski Zajazd	Варшава	Польша	
570				Польша Количество	7	
571		175.74			Польша Итог	
586		643.53			Португалия Итог	
589		29.46			Россия Итог	
713		13771.29			США Итог	
738		943.27			Финляндия Итог	
817		4274.99			Франция Итог	
837		1368.53			Швейцария Итог	
875		2992.81			Швеция Итог	
876				Общее количество	851	
877		64943.42			Общий итог	
878						
879						

Рис. 10.13. Полученные промежуточные итоги

Примечание

При добавлении в список промежуточных итогов разметка списка изменяется таким образом, что становится видна его структура. Нажимая знаки структуры [+], [−] и [1 2 3 4], можно создать итоговый отчет, скрыв подробности и отобразив только итоги.

Метод *Subtotal*

Метод `Subtotal` объекта `Range` добавляет промежуточные итоги в список данных, основываясь на изменениях в определенных полях данных. Промежуточные итоги позволяют обобщить данные. Метод `Subtotal` автоматически вставляет содержащие промежуточные итоги строки с формулами, по

которым подводятся итоги. Необходимо, чтобы до активизации этого метода данные были правильно отсортированы. В противном случае этот метод может привести к неверному выводу промежуточных итогов. Метод `Subtotal` программирует выполнение команды **Данные | Итоги**.

```
expression.Subtotal(GroupBy, Function, TotalList, Replace, PageBreaks,
↳SummaryBelowData)
```

- *expression* — ссылка на ячейку диапазона или на весь диапазон, для которого подводятся промежуточные итоги;
- *GroupBy* — обязательный параметр, задающий номер поля, по которому вычисляются промежуточные итоги;
- *Function* — обязательный параметр, определяющий функцию, по которой производится подсчет промежуточных итогов. Допустимыми значениями являются следующие константы `XlConsolidationFunction`: `xlAverage` (среднее арифметическое), `xlCount` (количество значений), `xlCountNums` (количество чисел), `xlMax` (максимум), `xlMin` (минимум), `xlProduct` (произведение), `xlStDev` (несмещенное отклонение), `xlStDevP` (смещенное отклонение), `xlSum` (сумма), `xlVar` (несмещенная дисперсия) и `xlVarP` (смещенная дисперсия);
- *TotalList* — обязательный параметр, специфицирующий массив целых чисел с номерами полей, по которым вычисляются промежуточные итоги;
- *Replace* — необязательный параметр, принимающий логические значения. Если его значение равно `True`, то существующие промежуточные итоги будут замещены;
- *PageBreaks* — необязательный параметр, принимающий логические значения. Если его значение равно `True`, то после каждой группы будет вставлено по символу разрыва страницы;
- *SummaryBelowData* — необязательный параметр, задающий местоположение вывода промежуточных итогов. Допустимыми значениями являются следующие константы `XlSummaryRow`: `xlSummaryAbove` (промежуточные итоги будут выведены над данными) и `xlSummaryBelow` (промежуточные итоги будут выведены под данными).

Удаление промежуточных итогов

Метод `RemoveSubtotal` объекта `Range` удаляет промежуточные итоги с рабочего листа. Например, следующая инструкция удаляет итоги, связанные с диапазоном **A1:I40**.

```
Range("A1:I40").RemoveSubtotal
```

Структура и объект *Outline*

В предыдущем разделе мы встретились со структурой. Программным представлением структуры является объект `Outline`. Давайте кратко обсудим, что такое структура, а также свойства и методы объекта `Outline`.

В общем случае структура — это такой режим работы, который позволяет разбить данные на определенные уровни детализации, просматривать только заголовки и подзаголовки документа. Структура наиболее полезна при создании итоговых отчетов, где не требуется чрезмерная детализация.

Следует помнить, что структуру невозможно создавать на основе произвольной таблицы данных. Ее построение основывается на таблицах с иерархической структурой данных.

Объект `Outline` инкапсулирует в себе данные о структуре. Свойство `Outline` рабочего листа возвращает объект `Outline`. На рабочем листе структуру можно создать командой **Данные | Группа и структура | Создание структуры**. Структура также появляется при построении промежуточных итогов и при консолидации данных. Удалить структуру можно командой **Данные | Группа и структура | Удалить структуру**. Приведем в табл. 10.4 основные свойства объекта `Outline`.

Таблица 10.4. *Основные свойства объекта* `Outline`

Свойство	Описание
`AutomaticStyles`	Принимает логические значения. Если значение этого свойства равно `True`, то структура строится на основе автоматических стилей
`SummaryColumn`	Возвращает местоположение итоговых столбцов. Допустимыми значениями являются следующие константы `XlSummaryColumn`: `xlLeft` (итоговые столбцы располагаются слева от столбцов, по которым подводятся итоги), `xlRight` (итоговые столбцы находятся справа)
`SummaryRow`	Возвращает местоположение итоговых строк. Допустимыми значениями являются следующие константы `XlSummaryRow`: `xlAbove` (итоговые строки располагаются выше строк, по которым подводятся итоги), `xlBelow` (итоговые строки располагаются ниже)

Отображение указанного числа уровней структуры

Объект `Outline` имеет единственный метод `ShowLevels`, который отображает указанное число уровней структуры по строкам и столбцам.

`ShowLevels(`*`RowLevels`*`, `*`ColumnLevels`*`)`

- *RowLevels* — необязательный параметр, устанавливающий число отображаемых уровней структуры по строкам;
- *ColumnLevels* — необязательный параметр, задающий число отображаемых уровней структуры по столбцам.

Удаление структуры

Метод `ClearOutline` объекта `Range` удаляет структуру. Например, следующая инструкция удаляет структуру, связанную с диапазоном **A1:I40**.

```
Range("A1:I40").ClearOutline
```

Отображение значков структуры

Метод `DisplayOutline` объекта `Window` принимает логические значения и управляет отображением значков структуры. Например, данная инструкция скрывает значки структуры.

```
ActiveWindow.DisplayOutline = False
```

Автоматическое создание структуры

Метод `AutoOutline` объекта `Range` автоматически создает структуру, которая заменяет уже существующую. Если объект `Range` является ячейкой, то структура создается для всего листа. Например, следующая инструкция создает структуру для диапазона **A1:I40**.

```
Range("A1:I40").AutoOutline
```

Пример приложения, подводящего промежуточные итоги и управляющего структурой

Попробуем применить метод `Subtotal` и объект `Outline` для решения несложной задачи. Воспользуемся учебной базой данных **Борей**. Возьмите из нее таблицу **Заказы**, точнее, поля **КодЗаказа**, **СтоимостьДоставки**, **НазваниеПолучателя**, **ГородПолучателя**, **СтранаПолучателя**, которую разместите на рабочем листе **Борей** (рис. 10.14).

Создайте форму, на ней расположите выключатель, счетчик, поле ввода и надпись. При установленном выключателе на нем будет отображаться надпись **Итоги подведены**, а на рабочем листе **Борей** с полями будут создаваться промежуточные итоги, подсчитывающие количество заказов, сделанных каждым из клиентов, и суммарные почтовые расходы по доставке этих заказов. Счетчик позволит управлять отображением различных уровней структуры промежуточных итогов. При снятом выключателе на нем отображается надпись **Итоги удалены**, а промежуточные итоги удаляются с рабоче-

го листа. Они также удаляются при закрытии диалогового окна. Итак, для завершения создания приложения в модуле формы наберите код из листинга 10.7.

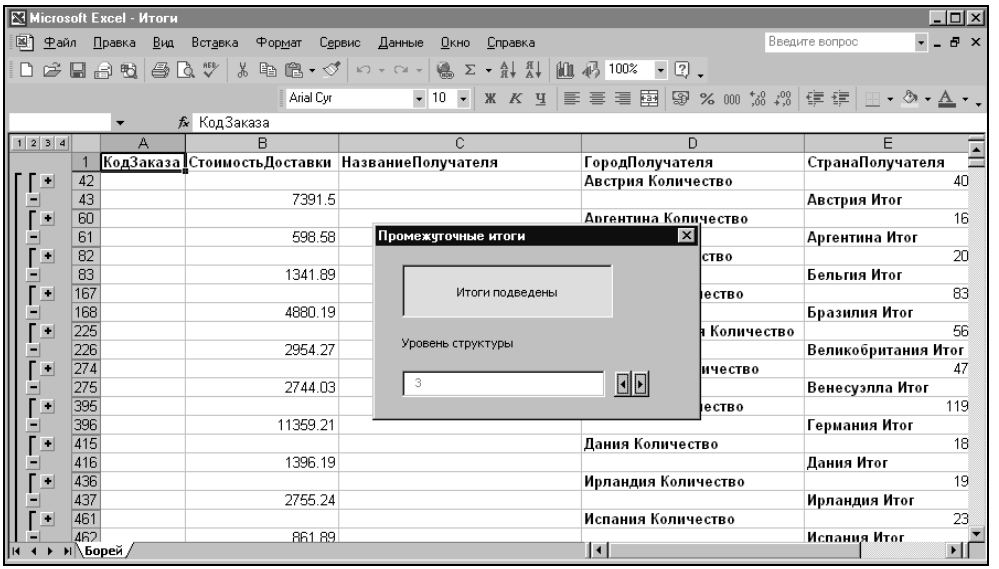


Рис. 10.14. Таблица **Заказы** и окно **Промежуточные итоги**

Листинг 10.7. Промежуточные итоги

```
Private Sub UserForm_Initialize()
   Me.Caption = "Промежуточные итоги"
   SpinButton1.Max = 4
   SpinButton1.Min = 1
   SpinButton1.Value = 4
   TextBox1.Enabled = False
   Label1.Caption = "Уровень структуры"
   ToggleButton1.Caption = "Итоги удалены"
   ToggleButton1.Value = False
End Sub

Private Sub UserForm_Terminate()
   DeleteSubtotal
   Columns("A:E").Select
   Selection.Columns.AutoFit
End Sub
```

```
Private Sub SpinButton1_Change()
   TextBox1.Text = SpinButton1.Value
   ToggleButton1.Value = True
   ActiveSheet.Outline.ShowLevels RowLevels:=SpinButton1.Value
   Columns("A:E").Select
   Selection.Columns.AutoFit
   Range("A1").Select
End Sub

Private Sub ToggleButton1_Change()
   If ToggleButton1.Value = True Then
      Rows("1:1").Select
      Selection.Sort Key1:=Range("C2"), Order1:=xlAscending
      Range("A2").Select
      Selection.Subtotal GroupBy:=5, Function:=xlSum, _
                TotalList:=Array(2), Replace:=False, _
                PageBreaks:=False, SummaryBelowData:=True
      Selection.Subtotal GroupBy:=5, Function:=xlCount, _
                TotalList:=Array(5), Replace:=False, _
                PageBreaks:=False, SummaryBelowData:=True
      ToggleButton1.Caption = "Итоги подведены"
      Range("A1").Select
   Else
      DeleteSubtotal
      ToggleButton1.Caption = "Итоги удалены"
   End If
End Sub

Private Sub DeleteSubtotal()
   Range("A2").Select
   Selection.RemoveSubtotal
   Rows("1:1").Select
   Selection.Sort Key1:=Range("A2"), Order1:=xlAscending
End Sub
```

Сценарии

Каждое уникальное значение в ячейке или каждая уникальная группа значений для группы ячеек называется *сценарием*. Сценарии позволяют проводить так называемый анализ данных "что, если". В ключевые ячейки можно вводить различные значения и смотреть, что при этом происходит. Довольно часто необходимо иметь под рукой различные варианты решения, а сценарии как раз и предоставляют эту возможность пользователю.

Объект *Scenario*

Объект `Scenario` позволяет хранить несколько значений в одной ячейке и представляет собой сценарий. Семейство `Scenarios` состоит из объектов `Scenario` и содержит в себе все сценарии рабочего листа. На рабочем листе вручную сценарии можно создать командой **Сервис | Сценарии**.

В табл. 10.5 перечислены наиболее важные методы семейства `Scenarios`.

Таблица 10.5. *Методы семейства* `Scenarios`

Метод	Описание
`Add`	Добавляет новый сценарий. `Add(`*`Name`*`, `*`ChangingCells`*`, `*`Values`*`, `*`Comment`*`, `*`Locked`*`, `*`Hidden`*`)` ♦ *`Name`* — имя сценария; ♦ *`ChangingCells`* — диапазон, отводимый под изменяемые ячейки сценария; ♦ *`Values`* — массив значений, вводимых в изменяемые ячейки; ♦ *`Comment`* — текстовая строка комментариев; ♦ *`Locked`* — свойство, принимающее логические значения. Если оно равно значению `True`, заблокировано изменение сценария; ♦ *`Hidden`* — свойство, принимающее логические значения. Если оно равно значению `True`, сценарий скрыт
`CreateSummary`	Довавляет в книгу новый рабочий лист, в котором создает отчет. `CreateSummary(`*`ReportType`*`, `*`ResultCells`*`)` ♦ *`ReportType`* — тип отчета. Допустимые значения: `xlStandardSummary` (стандартный отчет типа структуры), `xlSummaryPivotTable` (отчет в виде сводной таблицы); ♦ *`ResultCells`* — ссылка на ячейку или диапазон ячеек с формулами, которые зависят от значений из ячеек, указанных в параметре `ChangingCells` метода `Add`. Отчет создается на отдельном рабочем листе и не связан с исходными данными. Очень полезно перед составлением отчета присвоить имена ячейкам, заданным в параметре *`ResultCells`*. В противном случае вместо понятных имен в отчете будут помещены малопонятные ссылки на ячейки

В табл. 10.6 и 10.7 перечислены методы и свойства объекта `Scenario`.

Таблица 10.6. *Методы объекта* `Scenario`

Метод	Описание
`Show`	Показывает сценарий, вводя значения сценария в изменяемые ячейки
`Delete`	Удаляет сценарий
`ChangeScenario`	Изменяет группу изменяемых ячеек в сценарии. `ChangeScenario(`*`ChangingCells`*`, `*`Values`*`)` ♦ *`ChangingCells`* — группа ячеек, которая будет играть роль новой группы изменяемых ячеек; ♦ *`Values`* — массив с новыми значениями изменяемых ячеек

Таблица 10.7. *Свойства объекта* `Scenario`

Свойство	Описание
`ChangingCells`	Возвращает диапазон изменяемых ячеек. Например: `ActiveSheet.Scenarios(1).ChangingCells.Select`

`Values`	Возвращает массив текущих значений изменяемых ячеек. Например:
	`ActiveSheet.Scenarios(1).Values = Range("B1:B3")`
	или
	`ActiveSheet.Scenarios(1).Values = Array(1, 3, 5)`

Пример приложения по работе со сценариями

На простом примере покажем, как работает объект `Scenario`. Составим таблицу расходов ООО "Шарашкин и Компания" за январь месяц (рис. 10.15) и спрогнозируем расходы на следующий месяц. Свои прогнозы построим на основе предположения, что относительная величина расходов в феврале останется прежней, а их абсолютная величина увеличится с учетом инфляции. Будем считать, что инфляция в феврале будет в лучшем случае 1%, в худшем — 7%, а в наиболее вероятном случае — 3%.

Оформим работу со сценариями следующим образом:

1. На рабочем листе расположите один список, установите при помощи окна **Properties** значение его свойства `Name` равным `lstScenarios`. При открытии книги за счет обработки события `Open` объекта `Workbook` будет происходить заполнение списка названиями возможных сценариев инфляции в следующем месяце.

Щелчок на элементе списка приведет:

- к вводу значения уровня выбранного сценария инфляции в ячейку **E2**;
- к расчету предполагаемых расходов на следующий месяц.

2. В ячейку **B7** введите формулу, определяющую суммарные расходы:

```
=СУММ(B2:B6)
```

3. Выделите диапазон **C2:C7** и введите в него следующую формулу:

```
{=B2:B7*(1+E2)}
```

причем ее ввод надо завершить нажатием комбинации клавиш <Ctrl>+<Shift>+<Enter>, т. к. это формула массива. Такая формула позволяет определить ожидаемые расходы для целого диапазона значений.

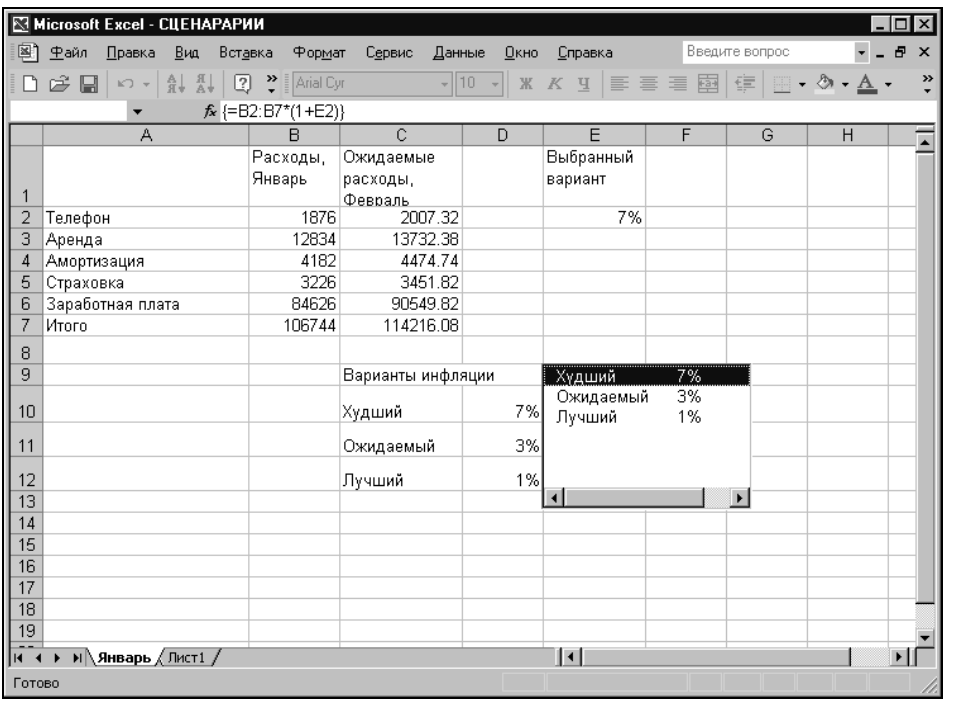


Рис. 10.15. Сценарии расходов

В коде такие сценарии будут иметь вид, представленный в листинге 10.8.

Листинг 10.8, *а*. Сценарии расходов на основе объекта `Scenario`. Модуль `ЭтаКнига`

```
Private Sub Workbook_Open()
   Dim sc As Scenario
```

```
   Dim i As Integer
   Dim V As Variant
   For Each sc In Worksheets("Январь").Scenarios
      sc.Delete
   Next
   With Worksheets("Январь")
      For i = 10 To 12
         V = .Cells(i, 4).Value
        .Scenarios.Add Name:=.Cells(i, 3).Value, _
                       ChangingCells:=.Range("E2"), _
                       Values:=V
      Next
      With .lstScenarios
         .ColumnCount = 2
         .ListFillRange = "C10:D12"
         .BoundColumn = 2
         .ListIndex = 0
     End With
   End With
End Sub
```

Листинг 10.8, *б*. Сценарии расходов на основе объекта `Scenario`. Модуль рабочего листа Январь

```
Private Sub lstScenarios_Click()
   Worksheets("Январь").Scenarios(lstScenarios.Text).Show
End Sub
```

Сценарии без объекта *Scenario*

В завершение обсуждения объекта `Scenario` стоит отметить, что команда **Сервис | Сценарии** и объект `Scenario` удобны при черновой подготовке проекта. В окончательной версии проекта сценарии более удобно и наглядно программировать непосредственно, без использования объекта `Scenario`. Например, только что рассмотренную задачу о прогнозировании расходов можно запрограммировать следующим образом (листинг 10.9), причем нет необходимости отводить на рабочем листе ячейку, в которую вводится выбранный вариант инфляции, и в диапазон **C2:C7** вручную вводить формулу.

Листинг 10.9, *а*. Сценарии расходов без объекта `Scenario`. Модуль `ЭтаКнига`

```
Private Sub Workbook_Open()
   With Worksheets("Январь").lstScenarios
      .ColumnCount = 2
```

```
      .ListFillRange = "C10:D12"
      .BoundColumn = 2
   End With
End Sub
```

Листинг 10.9, *б*. Сценарии расходов без объекта `Scenario`. Модуль рабочего листа Январь

```
Private Sub lstScenarios_Click()
   Range("C2:C7").FormulaArray = _
                  "=B2:B7*(1+" & lstScenarios.Value & ")"
End Sub
```

Сводная таблица

Сводные таблицы являются одним из наиболее мощных средств MS Excel по анализу баз данных, помещенных в таблицы или списки. Сводная таблица не просто группирует и обобщает данные, но и дает возможность провести глубокий анализ имеющейся информации. Создавая сводную таблицу, пользователь задает имена полей, которые размещаются в ее строках и столбцах. Допускается также задание поля страницы, которое позволяет работать со сводной таблицей, как со стопкой листов. Сводные таблицы удобны при анализе данных по нескольким причинам:

- позволяют создавать обобщающие таблицы, которые дают возможность группировать однотипные данные, подводить итоги, находить статические характеристики записей;
- легко преобразуются;
- позволяют производить автоматический отбор информации;
- на основе сводных таблиц строятся диаграммы, которые динамически перестраиваются вместе с изменениями сводной таблицы.

В последующих разделах на примере таблицы продаж, показанной на рис. 10.16, продемонстрируем, как создаются сводные таблицы и какие задачи можно решать с их помощью.

Подведение итогов по клиентам

Создадим сводную таблицу, которая определяет суммарную стоимость заказов для каждого клиента.

1. Выберите команду **Данные | Сводная таблица**. На экране появится первое окно мастера сводных таблиц (рис. 10.17).

	A	B	C	D	E
1	**Код заказа**	**Дата**	**Стоимость**	**Клиент**	**Страна**
2	10248	16.07.1996	32380	Wartian Herkku	Финляндия
3	10249	10.07.1996	11610	Toms Spezialitaten	Германия
4	10250	12.07.1996	65830	Hanari Carnes	Бразилия
5	10251	15.07.1996	41340	Victuailles en stock	Франция
6	10252	11.07.1996	51300	Supremes delices	Бельгия
7	10253	16.07.1996	58170	Hanari Carnes	Бразилия
8	10254	23.07.1996	22980	Chop-suey Chinese	Швейцария
9	10255	15.07.1996	148330	Richter Supermarkt	Швейцария
10	10256	17.07.1996	13970	Wellington Importadora	Бразилия
11	10257	22.07.1996	81910	HILARION-Abastos	Венесуэлла
12	10258	23.07.1996	140510	Ernst Handel	Австрия
13	10259	25.07.1996	3250	Centro comercial Moctezuma	Мексика
14	10260	29.07.1996	55090	Ottilies Kaseladen	Германия
15	10261	30.07.1996	3050	Que Delicia	Бразилия
16	10262	25.07.1996	48290	Rattlesnake Canyon Grocery	США
17	10263	31.07.1996	146060	Ernst Handel	Австрия
18	10264	23.08.1996	3670	Folk och fa HB	Швеция
19	10265	12.08.1996	55280	Blondel pere et fils	Франция
20	10266	31.07.1996	25730	Wartian Herkku	Финляндия
21	10267	06.08.1996	208580	Frankenversand	Германия
22	10268	02.08.1996	66290	GROSELLA-Restaurante	Венесуэлла
23	10269	09.08.1996	4560	White Clover Markets	США
24	10270	02.08.1996	136540	Wartian Herkku	Финляндия
25	10271	30.08.1996	4540	Split Rail Beer & Ale	США
26	10272	06.08.1996	98030	Rattlesnake Canyon Grocery	США
27	10273	12.08.1996	76070	QUICK-Stop	Германия
28	10274	16.08.1996	6010	Vins et alcools Chevalier	Франция
29	10275	09.08.1996	26930	Magazzini Alimentari Riuniti	Италия
30	10276	14.08.1996	13840	Tortuga Restaurante	Мексика
31	10277	13.08.1996	125770	Morgenstern Gesundkost	Германия

Продажи

Рис. 10.16. Таблица продаж

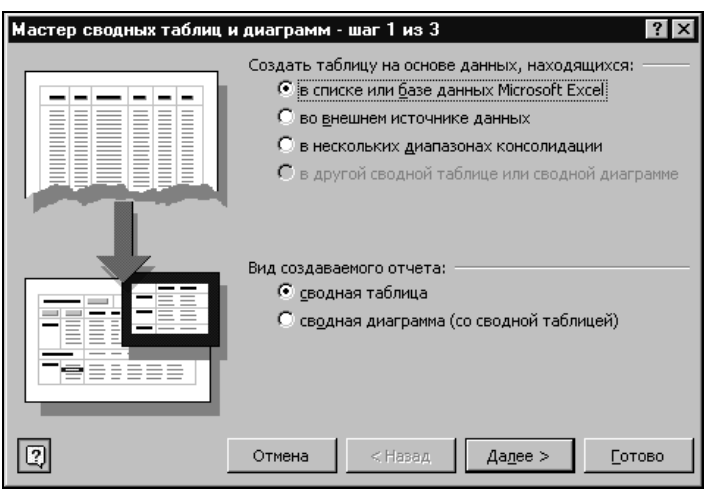


Рис. 10.17. Первое окно мастера сводных таблиц

2. В первом окне мастера сводных таблиц под заголовком **Создать таблицу на основе данных, находящихся** надо указать источник данных для создания сводной таблицы. Возможны четыре источника данных, которые выбираются при помощи одного из переключателей (табл. 10.8).

Таблица 10.8. *Переключатели источников данных*

Переключатель	Источник данных
в списке или базе данных Microsoft Excel	Список или таблица с помеченными столбцами, расположенная на рабочем листе
во внешнем источнике данных	Файлы и таблицы, созданные другими программами, например, Access
в нескольких диапазонах консолидации	Несколько списков или таблица с помеченными столбцами, расположенная на рабочем листе Excel
в другой сводной таблице или сводной диаграмме	Другая существующая в активной рабочей книге сводная таблица

В нашем случае выберите переключатель **в списке или базе данных Microsoft Excel**.

Кроме того, при помощи переключателей под заголовком **Вид создаваемого отчета** можно задать вид сводной таблицы — просто сводная таблица или сводная диаграмма со сводной таблицей. В нашем случае выберите переключатель **сводная таблица**. Нажмите кнопку **Далее**.

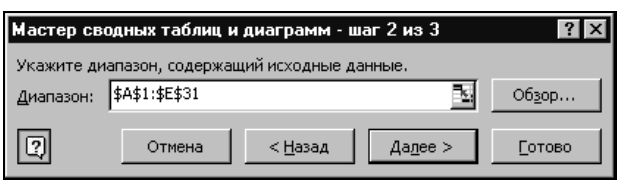


Рис. 10.18. Второе окно мастера сводных таблиц

3. На экране появится второе окно мастера сводных таблиц (рис. 10.18). На этом шаге вы должны указать диапазон, содержащий данные, по которым будет строиться сводная таблица. Если источник данных находится в другой рабочей книге, то необходимо воспользоваться кнопкой **Обзор**. Итак, после ввода ссылки на диапазон данных (в нашем случае `$A$1:$E$31`) нажмите кнопку **Далее**. На экране появится третье окно мастера сводных таблиц (рис. 10.19).
4. Первоначально для создания структуры сводной таблицы нажмите кнопку **Макет**. На экране отобразится окно **Макет** мастера сводных таблиц

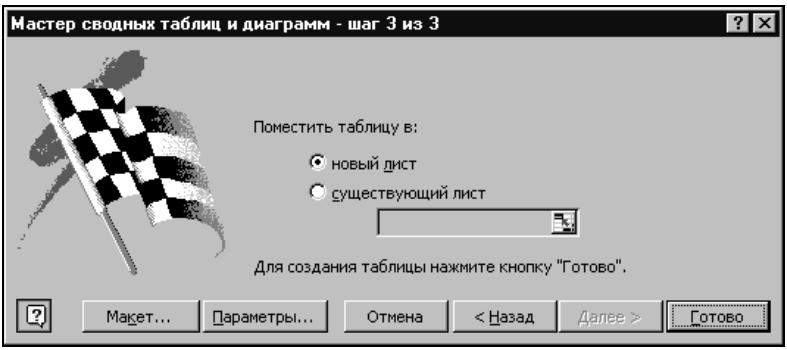


Рис. 10.19. Третье окно мастера сводных таблиц

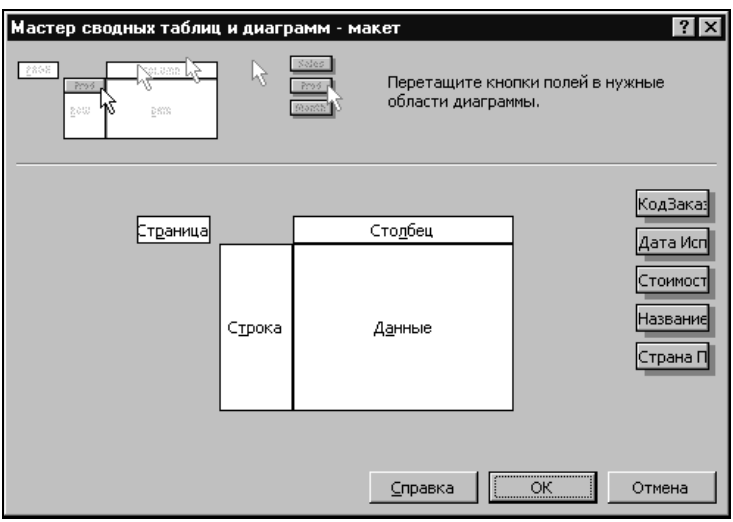


Рис. 10.20. Окно **Макет** мастера сводных таблиц

(рис. 10.20). Прежде чем создавать макет сводной таблицы, надо определиться, какая информация будет вводиться в области строк, столбцов, данных и страниц сводной таблицы.

- Итак, для создания структуры сводной таблицы выберите поле, содержащее данные, по которым нужно подвести итоги, и перетащите соответствующую кнопку в область **Данные**. В нашем случае будем подводить итоги по полю **Стоимость**. Для того чтобы выбрать операцию, по которой подводятся итоги, дважды щелкните по полю **Стоимость**, расположенной в области **Данные**. На экране отобразится окно **Вычисление поля сводной таблицы** (рис. 10.21). В списке **Операция** перечислены допустимые операции: Сумма, Количество, Среднее, Максимум, Минимум, Произведение, Количество чисел, Смещенное отклонение,

Несмещенное отклонение, Смещенная дисперсия, Несмещенная дисперсия. В нашем случае выберите Сумма. Список **Дополнительные вычисления** позволяет расширить множество допустимых операций. Его допустимые значения: Нет, Отличие, Доля, Приведенное отличие, С нарастающим итогом в поле, Доля от суммы по строке, Доля от суммы по столбцу, Доля от общей суммы, Индекс. Список **Дополнительные вычисления** отображается и скрывается последовательным нажатием кнопки **Дополнительно >>**. В нашем случае выберите в списке Нет и нажмите кнопку **OK**. Мы вернемся в окно **Макет**.

- Для того чтобы поместить элементы поля в строках с меткой в левой части таблицы, перетащите кнопку для выбранного поля в область **Строка**. В нашем случае по строкам будут размещаться клиенты, поэтому перетащите поле **Клиент** в область **Строка**.
- Для того чтобы поместить элементы поля в столбцах с меткой в верхней части таблицы, перетащите кнопку для выбранного поля в область **Столбец**. В нашем случае перетаскивать ничего не надо.
- Для того чтобы поместить элементы поля по страницам, что позволит работать со сводной таблицей как со стопкой листов, перетащите кнопку для выбранного поля в область **Страница**. В нашем случае перетаскивать ничего не надо.

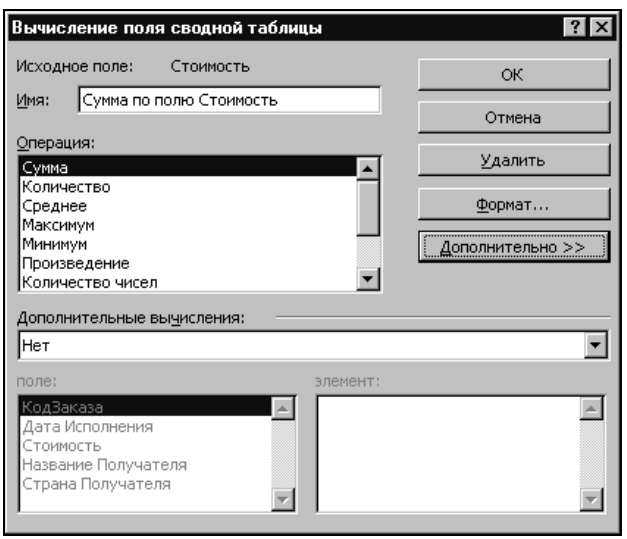


Рис. 10.21. Окно **Вычисление поля сводной таблицы**

Итак, в окне **Макет** мастера сводных таблиц создана структура будущей сводной таблицы (рис. 10.22). Нажмите на кнопку **OK**, и мы вернемся к третьему окну мастера сводных таблиц (см. рис. 10.19).

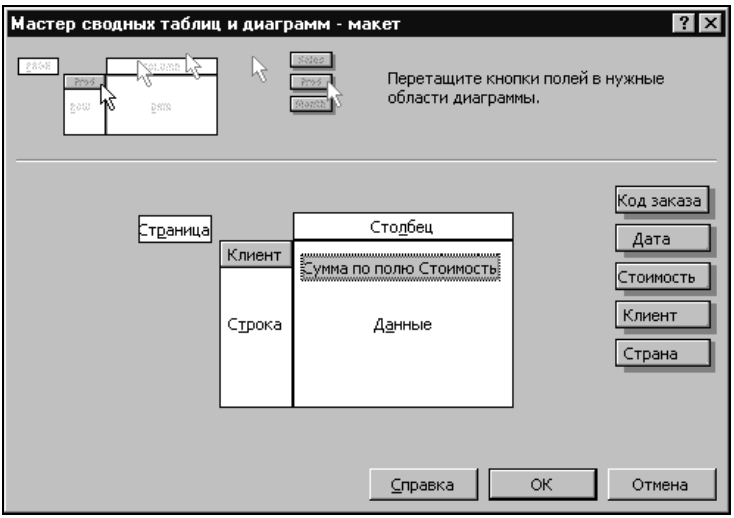


Рис. 10.22. Окно **Макет** со сконфигурированным макетом сводной таблицы

Примечание

Теперь можно позаботиться о параметрах сводной таблицы. Как правило, те значения параметров, которые установлены по умолчанию, являются оптимальными, и их не надо изменять. Но для того чтобы лучше разобраться с тем, как настраивается сводная таблица, нажмите кнопку **Параметры**. На экране отобразится окно **Параметры сводной таблицы** (рис. 10.23). В окне под полем **Имя,** в которое вводится имя сводной таблицы, находятся две группы параметров: **Формат** и **Данные**. Из группы **Формат** отметим только флажки **общая сумма по строкам** и **общая сумма по столбцам**, которые определяют, надо ли подводить итоги по строкам и столбцам. При построении сводной таблицы все данные копируются в скрытую *кэш-память*. Флажок **сохранить данные вместе с таблицей** определяет, будут ли в кэш-памяти при изменениях сводной таблицы сохраняться старые сведения или обновленные.

5. В группе **Поместить таблицу в** третьего окна мастера сводных таблиц (см. рис. 10.19) имеются два переключателя: **новый лист** и **существующий лист**, которые задают сводной таблице местоположение. Если выбран переключатель **существующий лист**, то в поле надо привести ссылку на верхнюю левую ячейку диапазона, где будет располагаться таблица. В нашем случае выберите переключатель **новый лист** и нажмите кнопку **Готово**. Сводная таблица построена (рис. 10.24).

Щелчок на кнопке ▼ в поле **Клиент** приводит к отображению списка с именами получателей. Этот список позволяет отображать информацию в сводной таблице как обо всех получателях, так и только о тех, которые выбраны.

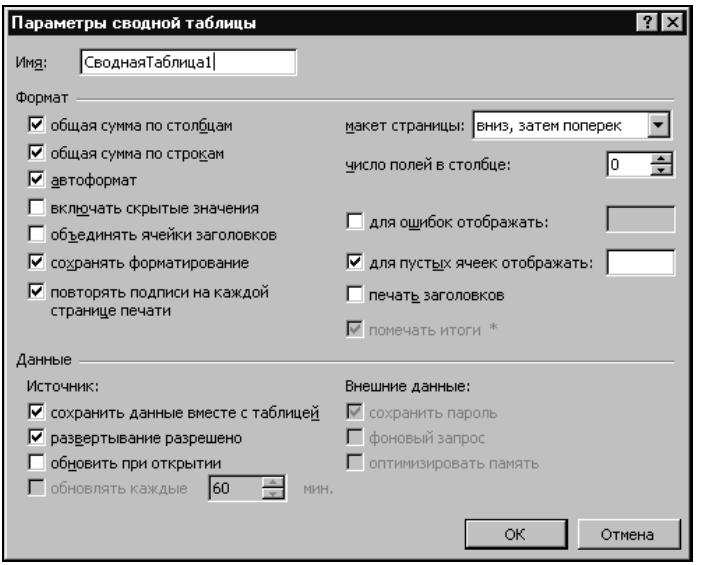


Рис. 10.23. Окно **Параметры сводной таблицы**

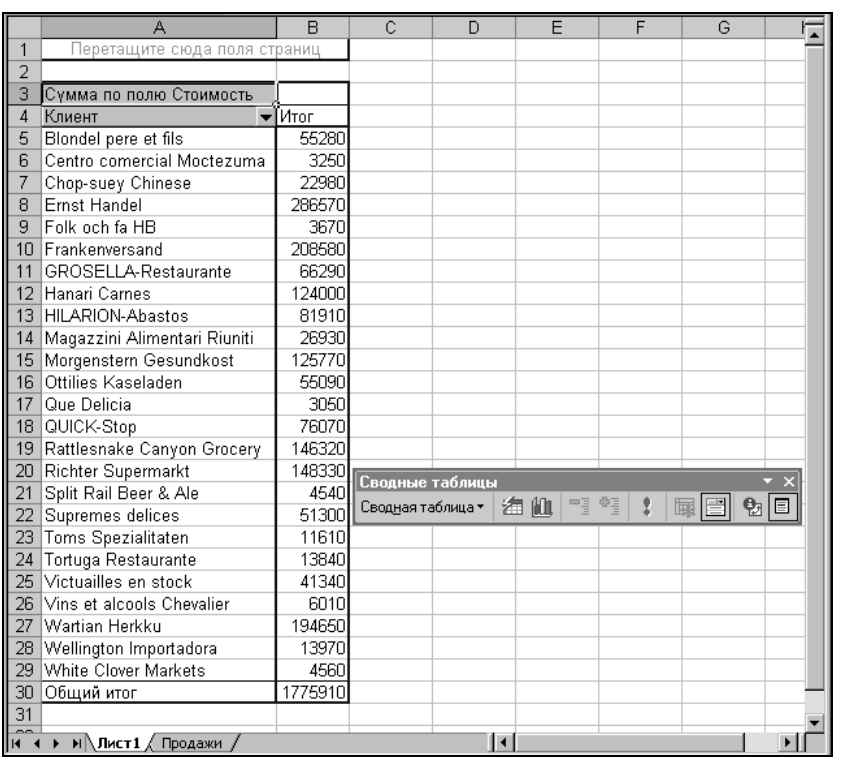


Рис. 10.24. Сводная таблица с панелью инструментов **Сводные таблицы**

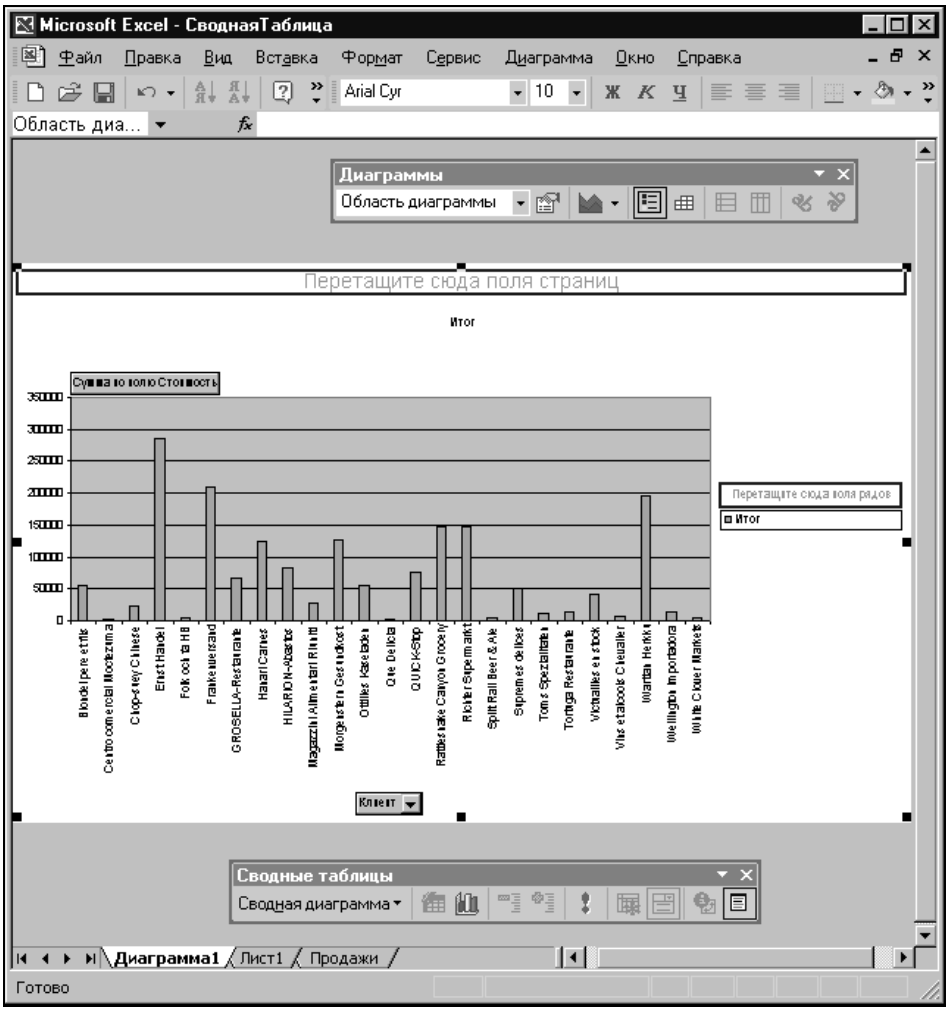


Рис. 10.25. Сводная диаграмма

Щелчок на кнопке **Мастер диаграмм** панели инструментов **Сводные таблицы** приводит к построению сводной диаграммы на базе сводной таблицы (рис. 10.25).

Обновление данных

Сводная таблица отображает содержание кэш-памяти, а не исходного диапазона данных. Если уже после создания сводной таблицы вы измените данные, на основе которых она была построена, то это не приведет к автоматическому изменению данных в сводной таблице. Сводная таблица не является динамической таблицей, автоматически обновляемой при обнов-

лении данных, на основе которых она построена. Для обновления сводной таблицы нужно выделить любую ячейку сводной таблицы и выбрать команду **Данные | Обновить данные** или нажать кнопку панели инструментов **Сводные таблицы**. При этом в кэш-память будут вновь занесены сведения, на основе которых строится сводная таблица.

Подведение итогов по клиентам и странам

Для подведения итогов по клиентам и странам в окне **Макет** мастера сводных таблиц перенесите поле **Страна** в область **Страница** (рис. 10.26).

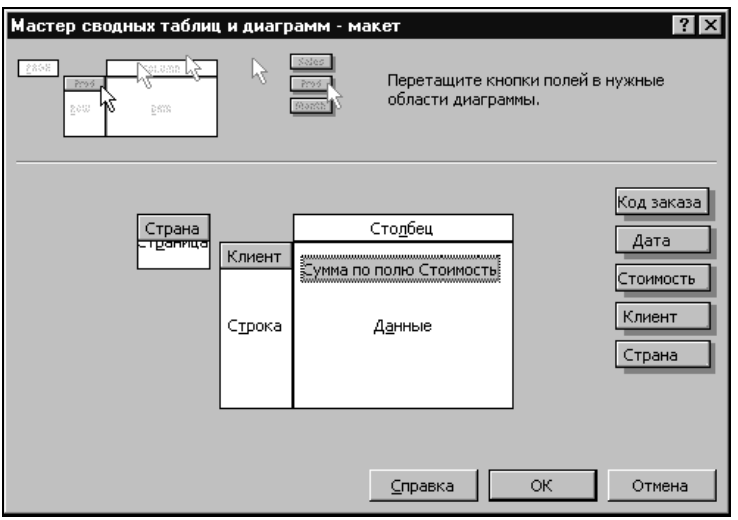


Рис. 10.26. Окно **Макет** мастера сводных таблиц при подведении итогов по клиентам и странам

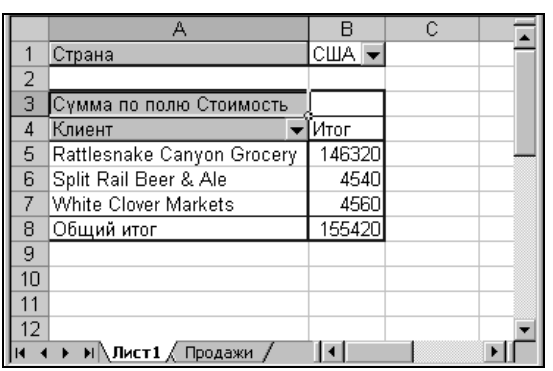


Рис. 10. 27. Результат подведения итогов по клиентам и странам

Результат подведения итогов при помощи сводной таблицы показан на рис. 10.27. Поле **Страна** позволяет отобразить данные как по отдельно выбранной стране, так и по всем странам.

Сортировка данных в сводной таблице

Для сортировки данных в сводной таблице, например, по стоимостям заказов клиентов в порядке убывания:

1. Выберите в сводной таблице поле **Клиент**.
2. В панели инструментов **Сводные таблицы** нажмите кнопку **Сводная таблица** и в появившемся контекстном меню выберите **Сортировка и десятка лучших**. На экране отобразится окно **Лучшая десятка и сортировка сводной таблицы** (рис. 10.28).

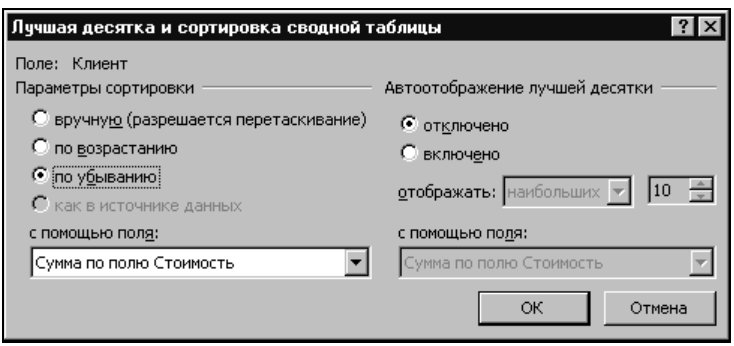


Рис. 10.28. Окно **Лучшая десятка и сортировка сводной таблицы**

3. В группе **Параметры сортировки** выберите переключатель **по убыванию**, а в списке **с помощью поля — Сумма по полю Стоимость**. Нажмите кнопку **ОК**. Результат сортировки сводной таблицы и соответствующая ей сводная диаграмма показаны на рис. 10.29.

Отображение заданного числа лучших или худших результатов

Для отображения в сводной таблице заданного числа лучших или худших результатов, например трех клиентов, сделавших наибольшие заказы:

1. Выберите в сводной таблице поле **Клиент**.
2. В панели инструментов **Сводные таблицы** нажмите кнопку **Сводная таблица** и в появившемся контекстном меню выберите **Сортировка и десятка лучших**. На экране отобразится окно **Лучшая десятка и сортировка сводной таблицы** (см. рис. 10.28). В группе **Автоотображение лучшей**

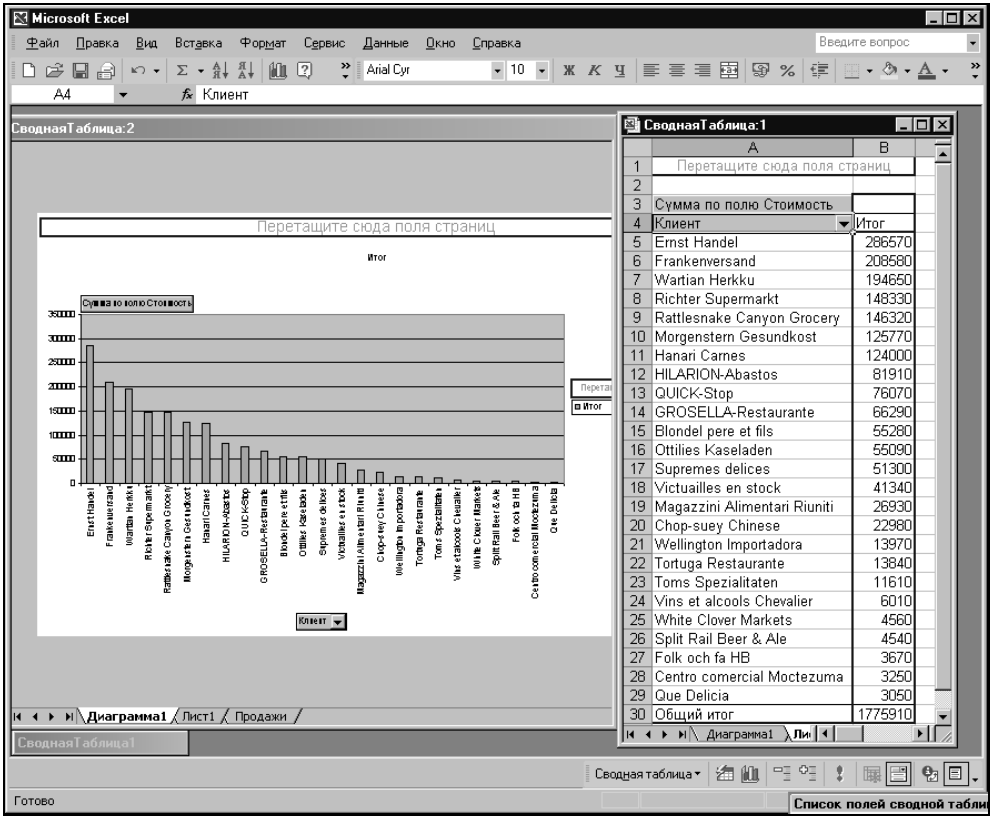


Рис. 10.29. Отсортированная по убыванию стоимости заказов клиентов сводная таблица и соответствующая сводная диаграмма

десятки выберите переключатель **включено**. В списке **отображать** выберите **наибольших**, а в соответствующее поле введите 3. Кроме того, для отображения отсортированных данных в группе **Параметры сортировки** выберите переключатель **по убыванию**, а в списке **с помощью поля — Сумма по полю Стоимость**. Нажмите кнопку **ОК**. Полученная преобразованная сводная таблица и соответствующая ей сводная диаграмма показаны на рис. 10.30.

Группировка данных по дням, месяцам, кварталам и годам

Сводная таблица позволяет группировать данные по дням, месяцам, кварталам и годам. Продемонстрируем это на примере группировки результатов продаж товаров клиентам по месяцам. Итак, в окне **Макет** мастера сводных таблиц перенесите как поле **Клиент**, так и поле **Дата** в область **Строка**

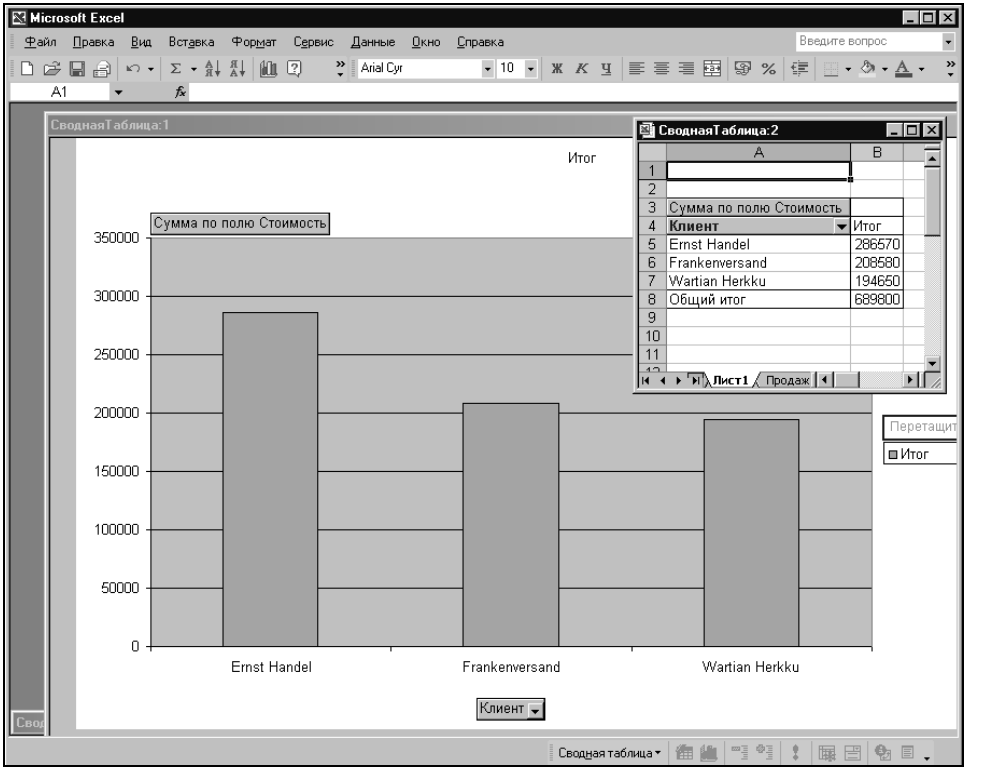


Рис. 10.30. Сводная таблица и соответствующая ей сводная диаграмма с отображением трех клиентов, сделавших наибольшие заказы

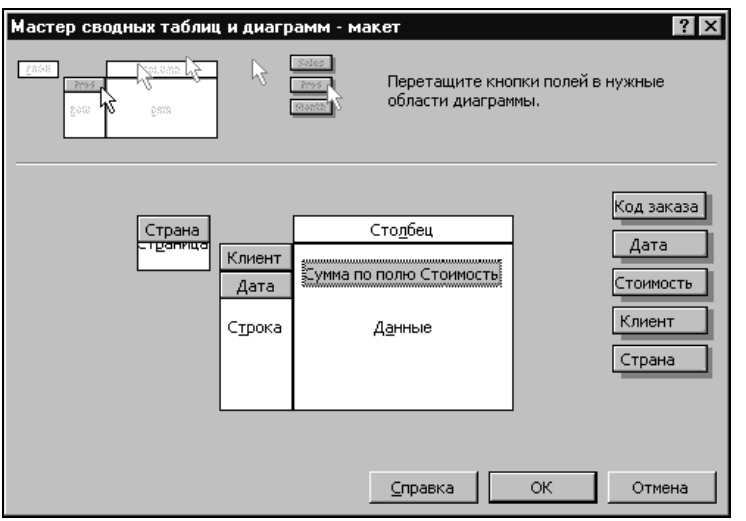


Рис. 10.31. Окно **Макет** мастера сводных таблиц при подведении итогов по клиентам и датам

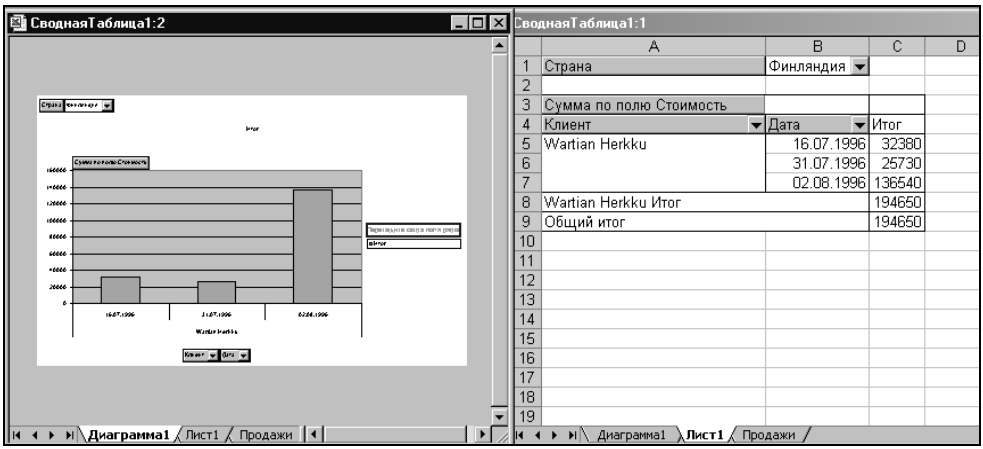


Рис. 10.32. Результат подведения итогов по клиентам и датам до группировки

(рис. 10.31). Результат подведения итогов при помощи сводной таблицы показан на рис. 10.32.

Для группировки данных по месяцам:

1. Щелкните на поле **Дата** правой кнопкой мыши. В появившемся контекстном меню выберите команду **Группа и структура | Группировать**.

2. На экране отобразится окно **Группирование** (рис. 10.33). В полях **начиная с** и **по** указываются границы временного интервала, в пределах которого производится группировка данных. В списке **с шагом** указывается шаг группировки. В нашем случае выберите **месяцы**. Нажмите кнопку **ОК**. Полученный результат группировки показан на рис. 10.34.

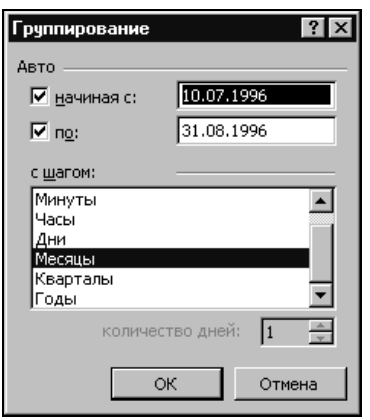


Рис. 10.33. Окно **Группирование**

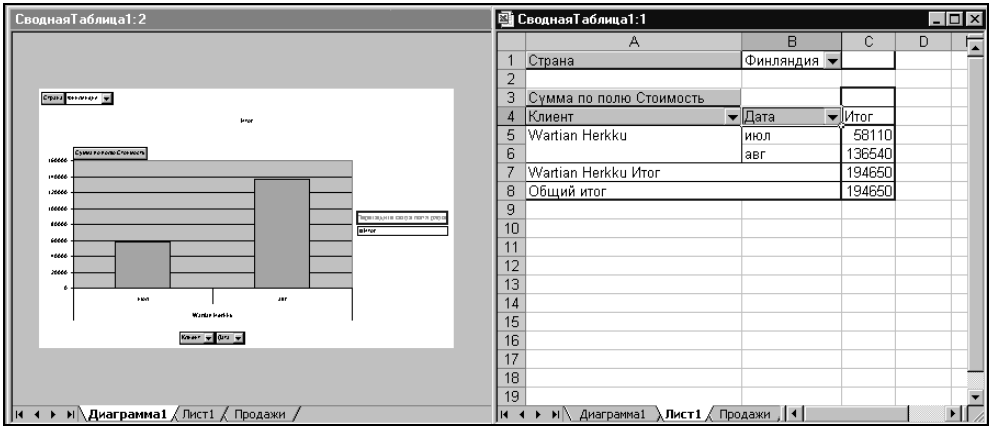


Рис. 10.34. Результат подведения итогов по клиентам и датам после группировки

Удаление группировки

Для удаления группировки выберите команду **Данные | Группа и структура | Разгруппировать**.

Удаление сводной таблицы

Для удаления сводной таблицы надо выбрать какую-нибудь ее ячейку. Затем в панели инструментов **Сводные таблицы** нажать кнопку **Сводная таблица** и в появившемся меню выбрать команду **Выделить | Таблицу целиком**, после чего выбрать команду **Правка | Очистить | Все**.

Нахождение как суммарной стоимости заказов каждого клиента, так и той доли, которую они занимают по отношению ко всем клиентам

Сводные таблицы позволяют производить вычисления по нескольким параметрам одновременно. Например, можно определить как суммарную стоимость заказов каждого клиента, так и ту долю, которую они занимают по отношению ко всем клиентам. Итак:

1. В окне **Макет** мастера сводных таблиц перенесите два поля **Стоимость** в область **Данные** (рис. 10.35).
2. Дважды щелкните по второму полю **Сумма по полю Стоимость**. На экране отобразится окно **Вычисление поля сводной таблицы**. В списке **Дополнительные вычисления** (который отобразится после щелчка на кнопке **Дополнительно>>**) выберите **Доля от общей суммы**. Нажмите кнопку **OK**.

Результат подведения итогов при помощи сводной таблицы показан на рис. 10.36.

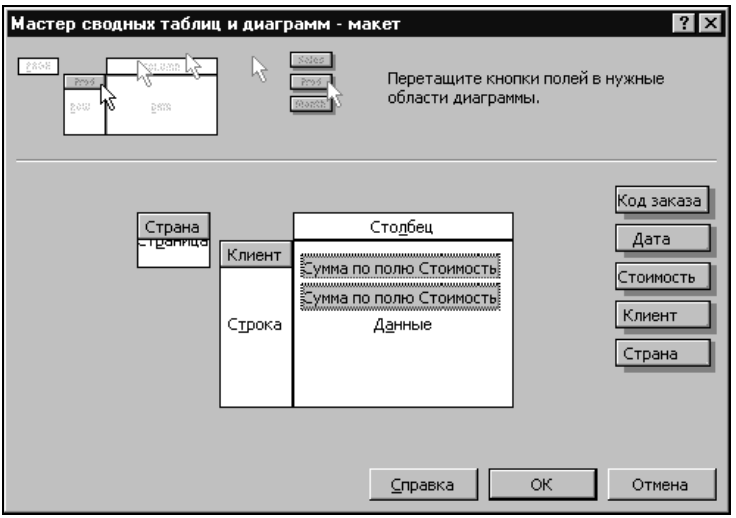


Рис. 10.35. Окно **Макет** мастера сводных таблиц

	A	B	C
1	Страна		
2			
3	Клиент	Данные	Итог
4	Rattlesnake Canyon Grocery	Сумма по полю Стоимость	146320
5		Сумма по полю Стоимость2	94.14%
6	Split Rail Beer & Ale	Сумма по полю Стоимость	4540
7		Сумма по полю Стоимость2	2.92%
8	White Clover Markets	Сумма по полю Стоимость	4560
9		Сумма по полю Стоимость2	2.93%
10	Итог Сумма по полю Стоимость		155420
11	Итог Сумма по полю Стоимость2		100.00%
12			
13			

Лист1 Продажи

Рис. 10.36. Нахождение как объемов заказов, так и их доли

Отображение детальных данных, содержащихся в ячейке сводной таблицы

Двойной щелчок на ячейке сводной таблицы вызывает добавление в книгу рабочего листа с таблицей, включающей все данные, содержащиеся в выбранной ячейке сводной таблицы.

Объекты, связанные со сводной таблицей

Со сводной таблицей связан ряд объектов, которые перечислены в табл. 10.9.

Таблица 10.9. *Объекты, связанные со сводной таблицей*

Объект	Описание
`PivotTable`	Сводная таблица
`PivotCache`	Кэш-память, отведенная под сводную таблицу. Этот объект возвращается методом `PivotCache` объекта `PivotTable`
`PivotCell`	Ячейка сводной таблицы. Данный объект можно получить при помощи свойства `PivotCell` объекта `Range`
`PivotField`	Поле сводной таблицы. Этот объект возвращается методом `PivotFields` объекта `PivotTable`
`PivotFormula`	Формула, по которой подводится итог в сводной таблице. Этот объект возвращается методом `PivotFormulas` объекта `PivotTable`
`PivotItem`	Элемент поля сводной таблицы. Этот объект возвращается методом `PivotItems` объекта `PivotTable`
`PivotLayout`	Данные о размещении полей и осей сводной таблицы и сводной диаграммы. Данный объект можно получить при помощи свойства `PivotLayout` объекта `Chart`

Все эти объекты являются членами соответствующих семейств, а именно `PivotTables`, `PivotTables`, `PivotFields`, `PivotFormulas`, `PivotItems` и `PivotItemList`.

Метод *PivotTableWizard*

Программно сводная таблица создается методом `PivotTableWizard` объектов `Worksheet`, `Workbook` или `PivotTable`. Вручную на рабочем листе сводная таблица конструируется при помощи мастера сводной таблицы, активизируемого командой **Данные | Сводная таблица**.

```
PivotTableWizard(SourceType, SourceData, TableDestination,
↳TableName, RowGrand, ColumnGrand, SaveData, HasAutoFormat, AutoPage,
↳Reserved, BackgroundQuery, OptimizeCache, PageFieldOrder,
↳PageFieldWrapCount, ReadData, Connection)
```

Все параметры этого метода являются необязательными.

- *SourceType* — тип источника данных. Допустимые значения: `xlConsolidation` (консолидация нескольких диапазонов рабочих листов), `xlDatabase` (спи-

сок или база данных MS Excel), `xlExternal` (внешняя база данных), `xlPivotTable` (сводная таблица);

- *SourceData* — определяет вид источника данных в зависимости от значения параметра *SourceType*. Если параметр *SourceType* принимает значение `xlDatabase`, то источником данных является диапазон. Если `xlExternal`, то массив строк, содержащий строку связи ODBC и SQL-оператор. Если `xlConsolidation`, то массив диапазонов. Если `xlPivotTable`, то имя существующей сводной таблицы;
- *TableDestination* — диапазон, где будет размещена сводная таблица;
- *TableName* — имя создаваемой сводной таблицы;
- *RowGrand* — параметр, принимающий логические значения. Если его значение равно `True`, то отображается суммарный итог по строкам сводной таблицы. Если его значение равно `False`, то итог не отображается;
- *ColumnGrand* — параметр, принимающий логические значения. Если его значение равно `True`, то отображается суммарный итог по столбцам сводной таблицы. Если его значение равно `False`, то итог не отображается;
- *SaveData* — параметр, принимающий логические значения. Если его значение равно `True`, то сохраняются данные вместе со сводной таблицей. Если его значение равно `False`, то сохраняется только сводная таблица;
- *HasAutoFormat* — параметр, принимающий логические значения. Если его значение равно `True`, то происходит автоматическое переформатирование сводной таблицы при изменении данных;
- *AutoPage* — применяется только при значении параметра `SourceType`, равном `xlConsolidation`. Допустимые значения `True` (MS Excel создает поле страницы) и `False` (пользователь должен создать поле);
- *Reserved* — не используется;
- *BackgroundQuery* — параметр, принимающий логические значения. Если его значение равно `True`, то MS Excel выполняет запрос в фоновом режиме, если `False` — в последовательном;
- *OptimizeCache* — параметр, принимающий логические значения. Если его значение равно `True`, то сводная таблица создается в режиме оптимизации, что применяется для сводных таблиц, обрабатывающих большие базы данных. Если его значение равно `False`, то оптимизация выключена, что убыстряет создание сводной таблицы;
- *PageFieldOrder* — задает последовательность, в которой поля добавляются в таблицу. Допустимые значения: `xlDownThenOver` и `xlOverThenDown`;
- *PageFieldWrapCount* — задает номер поля, с которого начинается новая страница. По умолчанию 0, т. е. отменена разбивка на страницы;

- *ReadData* — параметр, принимающий логические значения. Если его значение равно True, то данные сразу считываются в кэш. Если его значение равно False, то данные считываются в кэш по мере необходимости;
- *Connection* — используется для указания источника данных ODBC, источника данных URL и имени файла, содержащего запрос.

Тесно с методом PivotTableWizard связано свойство PivotTables объекта Worksheet, которое возвращает семейство сводных таблиц рабочего листа.

Объект *PivotTable*

Объект PivotTable инкапсулирует в себе данные о сводной таблице. Этот объект является членом семейства PivotTables. В табл. 10.10 приведены методы объекта PivotTable, а в табл. 10.11 — наиболее часто используемые свойства этого объекта.

Таблица 10.10. *Методы объекта* PivotTable

Метод	Описание
AddDataField	Добавляет поля с данными
AddFields	Добавляет строки, столбцы и страницы в сводную таблицу
CalculatedFields	Возвращает семейство CalculatedFields всех вычисляемых полей сводной таблицы
Format	Задает формат отчета сводной таблицы
GetData	Возвращает данные из указанной ячейки сводной таблицы
GetPivotData	Возвращает диапазон с информацией о сводной таблице
ListFormulas	Создает список формул на отдельном листе
PivotCache	Возвращает объект PivotCache
PivotFields	Возвращает семейство PivotFields
PivotSelect	Выбирает часть сводной таблицы
PivotTableWizard	Конструирует сводную таблицу по данной таблице
RefreshTable	Обновляет сводную таблицу. Для перерасчета сводной таблицы вручную надо ее выделить и выбрать команду **Данные | Обновить данные**
ShowPages	Устанавливает содержимое области **Страница**
Update	Обновляет связи в сводной таблице

Таблица 10.11. *Свойства объекта* `PivotTable`

Свойство	Описание
`ColumnFields`, `RowFields`, `DataFields`, `PageFields`	Возвращают объект (либо единичное поле, либо семейство полей), который является столбцом (строкой, данными и страницей) сводной таблицы
`VisibleFields`, `HiddenFields`	Возвращают объект, являющийся либо единичным полем, либо семейством полей, который в данный момент отображается (скрыт) в сводной таблице

Объект *PivotCache*

Объект `PivotCache` представляет собой кэш-память, выделенную под конкретную сводную таблицу. Этот объект является членом семейства `PivotCaches` и возвращается методом `PivotCache` объекта `PivotTable`.

Основным методом семейства `PivotCaches` является метод `Add`. Он имеет следующий синтаксис:

```
Add(SourceType, SourceData)
```

- *`SourceType`* — обязательный параметр, задающий тип данных, на основе которых строится сводная таблица. Допустимыми значениями являются следующие константы `XlPivotTableSourceType`: `xlConsolidation` (консолидация нескольких диапазонов рабочих листов), `xlDatabase` (список или база данных MS Excel), `xlExternal` (внешняя база данных), `xlPivotTable` (сводная таблица);
- *`SourceData`* — необязательный параметр, определяющий вид источника данных в зависимости от значения параметра *`SourceType`*. Этот параметр является обязательным, если значение параметра *`SourceType`* отлично от `xlExternal`.

Объект `PivotCache` имеет ряд методов, перечисленных в табл. 10.12.

Таблица 10.12. *Методы объекта* `PivotCache`

Метод	Описание
`CreatePivotTable`	Создает объект `PivotTable`. `CreatePivotTable(TableDestination, TableName, ReadData)` ♦ *`TableDestination`* — необязательный параметр, дающий ссылку на ячейку, в которой будет располагаться верхний левый угол сводной таблицы. Если сводная таблица создается на новом рабочем листе, то значение этого параметра должно быть равно пустой строке;

Таблица 10.12 (окончание)

Метод	Описание
(прод.)	♦ *TableName* — необязательный параметр, задающий имя сводной таблицы; ♦ *ReadData* — необязательный параметр, принимающий логические значения и определяющий, надо ли сводную таблицу строить по всей внешней базе данных
MakeConnection	Устанавливает соединения с кэш-памятью
Refresh	Обновляет кэш-память
ResetTimer	Переустанавливает таймер обновления кэш-памяти
SaveAsODC	Запись кэш-памяти в файл ODC (Microsoft Office Data Connection)

Объект *PivotField*

Объект PivotField представляет собой поле сводной таблицы. Этот объект является членом семейства PivotFields и возвращается методом PivotField объекта PivotTable. В табл. 10.13 перечислены методы, а в табл. 10.14 — основные свойства объекта PivotField.

Таблица 10.13. *Методы объекта* PivotField

Метод	Описание
AddPageItem	Добавляет элемент в поле
AutoShow	Задает правило автоматического отображения элементов полей
AutoSort	Задает правило автоматической сортировки полей
CalculatedItems	Возвращает семейство CalculatedItems
Delete	Удаляет поле
PivotItems	Возвращает семейство PivotItems всех элементов поля

Таблица 10.14. *Основные свойства объекта* PivotField

Свойство	Описание
Orientation	Возвращает местоположения поля в сводной таблице. Допустимые значения: xlColumnField, xlDataField, xlHidden, xlPageField и xlRowField

Таблица 10.14 *(окончание)*

Свойство	Описание
`Name`	Имя поля
`Function`	Функция, по которой в поле подводится итог. Допустимые значения: `xlAverage`, `xlCount`, `xlCountNums`, `xlMax`, `xlMin`, `xlProduct`, `xlStDev`, `xlStDevP`, `xlSum`, `xlVar` и `xlVarP`
`Position`	Возвращает позицию поля (первое, второе и т. д.) среди полей того же местоположения

Пример приложения, помогающего построить и обновлять сводную таблицу

Приведем простое приложение, которое автоматизирует процесс построения, обновления и удаления сводной таблицы и сводной диаграммы. Автоматизация построения сводной таблицы в ситуации, когда список заказов постоянно обновляется, бесспорно, является хорошим подспорьем.

Итак, в проекте имеются три листа:

- рабочий лист **Продажи** (см. рис. 10.16) — на этом листе расположена таблица, на базе которой будут строиться сводная таблица и диаграмма. Данная таблица состоит из пяти полей: **Код заказа**, **Дата**, **Стоимость**, **Клиент**, **Страна**. Заголовки полей расположены в ячейках **A1**, **B1**, **C1**, **D1**, **E1**. Число полей произвольно и может изменяться со временем;
- рабочий лист **Отчет** — на нем создается сводная таблица;
- лист диаграмм **Диаграмма** — на нем конструируется сводная диаграмма. Этот лист необязательный. В случае его отсутствия он автоматически будет создан при построении сводной диаграммы.

Программа данного приложения разбита на два листинга. В листинге 10.10 собраны процедуры, обеспечивающие функционирование бизнес-логики проекта, а в листинге 10.11 — процедуры, создающие его интерфейс, который представляет собой панель инструментов, состоящую из трех кнопок:

- кнопка **Сводная таблица** — конструирует сводную таблицу и диаграмму, причем автоматически определяется число записей в таблице на листе **Отчет**;
- кнопка **Удаление** — удаляет сводную таблицу и диаграмму без удаления рабочего листа **Отчет** и листа диаграмм **Диаграмма**;
- кнопка **Обновление** — обновляет сводную таблицу и диаграмму. В случае, если число записей в таблице на листе **Отчет** было изменено, то для об-

новления сводной таблицы ее надо сначала удалить, нажав кнопку **Удаление**, а затем заново построить, нажав кнопку **Сводная таблица**.

Опишем функции процедур, используемых в листинге 10.10:

- функция `CheckSheet` реализует проверку наличия специфицированного рабочего листа в проекте;
- процедура `GetRange` определяет, на основе какого диапазона листа **Продажи** конструируется сводная таблица;
- процедура `PivotTblBuilder` проверяет при помощи функции `CheckSheet` наличие рабочего листа **Отчет** и, в случае его отсутствия, информирует пользователя о необходимости создания такового. Затем она проверяет наличие сводной таблицы. В случае ее наличия, она также информирует пользователя. Если же лист **Отчет** существует, а сводной таблицы нет, то она конструирует таковую по диапазону, возвращаемому процедурой `GetRange`;
- процедура `RefreshPivotTbl` обновляет сводную таблицу;
- процедура `DeletePivotTbl` удаляет сводную таблицу.

Листинг 10.10. Построение и обновление сводной таблицы. Стандартный модуль

```
Option Explicit

Public nameCB As String
Public PivotTblName As String
Public PivotTblSheetName As String
Public SheetName As String
Public ChartSheetName As String

Sub PivotTblBuilder()
   Dim rgn As String
   Dim PivotTblAddress As String

   If Not CheckSheet(SheetName) Then Exit Sub
   If Not CheckSheet(PivotTblSheetName) Then Exit Sub
   If Worksheets(PivotTblSheetName).PivotTables.Count > 0 Then
      Dim pt As PivotTable
      For Each pt In Worksheets(PivotTblSheetName).PivotTables
          If pt.Name = PivotTblName Then
             MsgBox "Сводная таблица уже существует"
          Exit Sub
          End If
```

```
      Next
   End If

   PivotTblAddress = "[" & ActiveWorkbook.Name & "]" _
                     & PivotTblSheetName & "!R3C1"
   GetRange SheetName, rgn
   Worksheets(SheetName).Activate
   Range("A1").Select
   ActiveWorkbook.PivotCaches.Add(SourceType:=xlDatabase, _
                                  SourceData:=rgn).CreatePivotTable _
      TableDestination:=PivotTblAddress, _
      TableName:=PivotTblName, _
      DefaultVersion:=xlPivotTableVersion10
   Worksheets(PivotTblSheetName).PivotTables(1).AddFields _
                                  RowFields:="Клиент", _
                                  PageFields:="Страна"
   Worksheets(PivotTblSheetName).PivotTables(PivotTblName). _
         PivotFields("Стоимость").Orientation = xlDataField

   Dim c As Chart
   For Each c In Charts
     If c.Name = ChartSheetName Then
        Application.DisplayAlerts = False
        c.Delete
        Application.DisplayAlerts = True
        Exit For
     End If
   Next
   Charts.Add
   ActiveChart.Name = ChartSheetName
   ActiveChart.SetSourceData _
       Source:=Sheets(PivotTblSheetName).Range("A3")
   ActiveChart.Location Where:=xlLocationAsNewSheet
End Sub

Private Function CheckSheet(ShName As String) As Boolean
   Dim ws As Worksheet
   Dim flag As Boolean
   flag = False
   For Each ws In Worksheets
      If ws.Name = ShName Then
        flag = True
        Exit For
      End If
```

```
   Next
   If Not flag Then MsgBox "В книге должен находиться лист " & ShName
   CheckSheet = flag
End Function

Private Sub GetRange(ByRef SheetName As String, ByRef rgn As String)
   Dim r As Range
   Set r = Worksheets(SheetName).Range("A1").CurrentRegion
   rgn = SheetName & "!" & r.Address(ReferenceStyle:=xlR1C1)
End Sub

Sub RefreshPivotTbl()
   Dim ptbl As PivotTable
   ActivatePivotTblSheet
   If ActiveSheet.Name = PivotTblSheetName Then
      For Each ptbl In ActiveSheet.PivotTables
         If ptbl.Name = PivotTblName Then
            ActiveSheet.PivotTables(PivotTblName).RefreshTable
           Exit Sub
         End If
      Next
   End If
End Sub

Sub DeletePivotTbl()
   Dim ptbl As PivotTable
   ActivatePivotTblSheet
   If ActiveSheet.Name = PivotTblSheetName Then
      For Each ptbl In ActiveSheet.PivotTables
         ptbl.TableRange2.Clear
      Next
   End If
End Sub

Private Sub ActivatePivotTblSheet()
   Dim ws As Worksheet
   For Each ws In Worksheets
      If ws.Name = PivotTblSheetName Then
        ws.Activate
      End If
   Next
End Sub
```

Примечание

Обратите внимание на то, как в программе удаляется сводная таблица с рабочего листа. Первоначально к объекту `PivotTable` (сводной таблице) применяется свойство `TableRange2`, которое возвращает объект `Range` — диапазон, содержащий все области, составляющие сводную таблицу. Сводная же таблица удаляется применением метода `Clear` к полученному объекту `Range`. Кроме свойства `TableRange2`, объект `PivotTable` обладает следующими полезными свойствами, возвращающими объект `Range`:

- `ColumnRange` — диапазон, содержащий область **Столбец**;
- `RowRange` — диапазон, содержащий область **Строка**;
- `PageRange` — диапазон, содержащий область **Страница**;
- `DataBodyRange` — диапазон, содержащий область **Данные**;
- `DataLabelRange` — диапазон, содержащий имя поля сводной таблицы в области **Данные**;
- `TableRange1` — диапазон, содержащий области **Строка**, **Столбец** и **Данные**.

Листинг 10.11. Построение и обновление сводной таблицы. Модуль ЭтаКнига

```
Private Sub Workbook_Open()
   nameCB = "МояСводнаяТаблица"
   PivotTblName = "СводнаяТаблицаПродажи"
   PivotTblSheetName = "Отчет"
   SheetName = "Продажи"
   ChartSheetName = "Диаграмма"

   Dim cb As CommandBar
   Dim ccbBuilder As CommandBarButton
   Dim ccbRefresh As CommandBarButton
   Dim ccbDelete As CommandBarButton

   Set cb = Application.CommandBars.Add(Name:=nameCB, _
                                        Position:=msoBarTop, _
                                        MenuBar:=False, _
                                        Temporary:=True)
   With cb
      .Visible = True
      .Protection = msoBarNoMove + msoBarNoChangeVisible _
                    + msoBarNoCustomize
      With .Controls
         Set ccbBuilder = .Add(Type:=msoControlButton, _
                               Temporary:=True)
```

```
        With ccbBuilder
           .Style = msoButtonCaption
           .Caption = "Сводная таблица"
           .OnAction = "PivotTblBuilder"
        End With

        Set ccbRefresh = .Add(Type:=msoControlButton, Temporary:=True)
        With ccbRefresh
           .Style = msoButtonCaption
           .Caption = "Обновление"
           .OnAction = "RefreshPivotTbl"
        End With

        Set ccbDelete = .Add(Type:=msoControlButton, Temporary:=True)
        With ccbDelete
           .Style = msoButtonCaption
           .Caption = "Удаление"
           .OnAction = "DeletePivotTbl"
        End With
     End With
  End With
End Sub

Private Sub Workbook_BeforeClose(Cancel As Boolean)
   Application.CommandBars(nameCB).Delete
End Sub
```

Поиск решения

Поиск решения (solver) — это мощное средство MS Excel, позволяющее в общем случае решать задачи как линейного, так и нелинейного программирования. Как частный случай, поиск решения позволяет решать и те задачи, которые решаются при помощи подбора параметра.

Примечание

Команда **Поиск решения** находится в меню **Сервис**. Если ее там нет, то средство Поиск решения необходимо установить. Для этого:

- выберите команду **Надстройки** в меню **Сервис**. Если в списке диалогового окна **Надстройки** отсутствует пункт **Поиск решения**, то повторно запустите программу установки Microsoft Office и убедитесь, что выбран параметр установки надстройки **Поиск решения**.
- в диалоговом окне **Надстройки** установите флажок **Поиск решения** и нажмите кнопку **OK**. Если на экране появится сообщение с вопросом о том, следует ли установить соответствующее средство, нажмите кнопку **Да**.

Решение транспортной задачи при помощи средства Поиск решения

Рассмотрим следующую простую задачу. Фирма MIS, Inc. имеет 4 фабрики и 5 центров распределения ее товаров. Фабрики располагаются в Денвере, Бостоне, Нью-Орлеане и Далласе с производственными возможностями соответственно в 200, 150, 225 и 175 единиц продукции ежедневно. Распределительные центры находятся в Лос-Анджелесе, Далласе, Сент-Луисе, Вашингтоне и Атланте с потребностями соответственно в 100, 200, 50, 250 и 150 единиц продукции ежедневно. Хранение на фабрике не поставленной в центр распределения единицы продукции обходится в $0,75 в день, а штраф за просрочку поставки заказанной потребителем в центре распределения единицы продукции, но там не находящейся, равен $2,5 в день. Стоимость перевозки единицы продукции с фабрик в пункты распределения приведена в табл. 10.15.

Таблица 10.15. *Транспортные расходы, $*

	1 Лос-Анджелес	2 Даллас	3 Сент-Луис	4 Вашингтон	5 Атланта
1 Денвер	1,5	2	1,75	2,25	2,25
2 Бостон	2,5	2	1,75	1	1,5
3 Нью-Орлеан	2	1,5	1,5	1,75	1,75
4 Даллас	2	0,5	1,75	1,75	1,75

Необходимо так спланировать перевозки, чтобы минимизировать суммарные транспортные расходы.

Важно отметить, что т. к. данная модель сбалансирована, т. е. суммарный объем произведенной продукции равен суммарному объему потребностей в данной продукции, то в этой модели не надо учитывать издержки, связанные как со складированием, так и с недопоставками продукции. В противном случае в модель надо ввести:

- в случае перепроизводства — фиктивный пункт распределения; стоимость перевозок единицы продукции в этот фиктивный пункт полагается равной стоимости складирования, а объемы перевозок в этот пункт равны объемам складирования излишков продукции на фабриках;
- в случае дефицита — фиктивную фабрику; стоимость перевозок единицы продукции из фиктивной фабрики полагается равной стоимости штрафов за недопоставку продукции, а объемы перевозок из этой фабрики равны объемам недопоставок продукции в пункты распределения.

Для решения этой задачи построим ее математическую модель. Неизвестными в задаче являются объемы перевозок. Пусть x_{ij} — объем перевозок с i-й фабрики в j-й центр распределения. Функцией цели являются суммарные транспортные расходы, т. е.

$$z = \sum_{i=1}^{4} \sum_{j=1}^{5} c_{ij} x_{ij},$$

где c_{ij} — стоимость перевозки единицы продукции с i-й фабрики в j-й центр распределения. Кроме того, неизвестные должны удовлетворять следующим ограничениям:

- неотрицательность объема перевозок;
- т. к. модель сбалансирована, то вся продукция должна быть вывезена с фабрик и потребность всех центров распределения должна быть полностью удовлетворена.

Таким образом, мы имеем следующую модель:

Минимизировать:

$$z = \sum_{i=1}^{4} \sum_{j=1}^{5} c_{ij} x_{ij},$$

при ограничениях:

$$\sum_{j=1}^{5} x_{ij} = a_i,\ i \in [1,\ 4],$$

$$\sum_{i=1}^{4} x_{ij} = b_j,\ j \in [1,\ 5],$$

$$x_{ij} \geq 0,\ i \in [1,\ 4],\ j \in [1,\ 5].$$

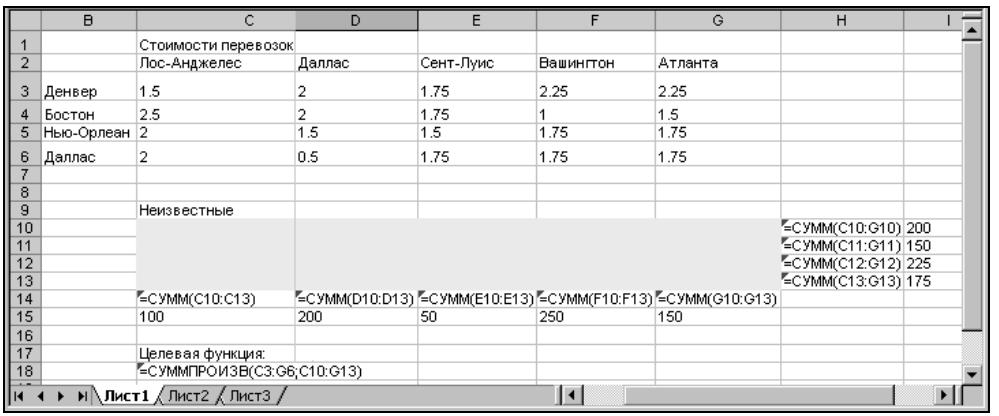
где a_i — объем производства на i-й фабрике, b_j — спрос в j-м центре распределения.

Для решения этой задачи с помощью средства Поиск решения введите данные в рабочий лист, как показано на рис. 10.37. В диапазон ячеек **C3:G6** введите стоимости перевозок. Диапазон ячеек **C10:G13** отведен под значения неизвестных (объемы перевозок). В диапазон ячеек **I10:I13** введите объемы производства на фабриках, а в диапазон ячеек **C15:G15** — объемы потребности в продукции в пунктах распределения. В ячейку **C18** введите функцию цели

```
=СУММПРОИЗВ(C3:G6;C10:G13)
```

В ячейки диапазона **H10:H13** введите формулы, возвращающие суммарные объемы продукции, вывозимые с каждой из фабрик. В ячейки диапазона

C14:G14 — формулы, возвращающие суммарные объемы продукции, ввозимой в каждый из центров распределения.



	B	C	D	E	F	G	H	I
1		Стоимости перевозок						
2		Лос-Анджелес	Даллас	Сент-Луис	Вашингтон	Атланта		
3	Денвер	1.5	2	1.75	2.25	2.25		
4	Бостон	2.5	2	1.75	1	1.5		
5	Нью-Орлеан	2	1.5	1.5	1.75	1.75		
6	Даллас	2	0.5	1.75	1.75	1.75		
7								
8								
9		Неизвестные						
10							=СУММ(C10:G10)	200
11							=СУММ(C11:G11)	150
12							=СУММ(C12:G12)	225
13							=СУММ(C13:G13)	175
14		=СУММ(C10:C13)	=СУММ(D10:D13)	=СУММ(E10:E13)	=СУММ(F10:F13)	=СУММ(G10:G13)		
15		100	200	50	250	150		
16								
17		Целевая функция:						
18		=СУММПРОИЗВ(C3:G6;C10:G13)						

Лист1 Лист2 Лист3

Рис. 10.37. Исходные данные транспортной задачи

Переходим к нахождению оптимального объема перевозок фабриками и центрами распределения:

1. Выберите команду **Сервис | Поиск решения**. На экране отобразится диалоговое окно **Поиск решения** (рис. 10.38). Окно **Поиск решения** имеет элементы, перечисленные в табл. 10.16.

Таблица 10.16. *Элементы окна* ***Поиск решения***

Элемент	Описание
Поле **Установить целевую ячейку**	Приводится ссылка на ячейку с функцией, максимум, минимум или значение которой Поиск решения будет искать, изменяя значения параметров так, чтобы они удовлетворяли налагаемым на них ограничениям. В примере с транспортной задачей в поле **Установить целевую ячейку** введите `C18`
Группа **Равной**	Тип взаимосвязи между решением и целевой ячейкой устанавливается путем выбора переключателя в группе **Равной**. Для отыскания максимального значения целевой функции выбирается переключатель **максимальному значению**, минимального — переключатель **минимальному значению**. Если отыскиваются значения переменных, для которых значение функции из целевой ячейки равно установленному в поле **значению** группы **Равной**, то выбирается переключатель **значению**. В транспортной задаче выберите переключатель **минимальному значению**, т. к. разрабатывается план перевозок с минимальными транспортными расходами

Таблица 10.16 *(окончание)*

Элемент	Описание
Поле **Изменяя ячейки**	Приводится ссылка на диапазон ячеек или группу диапазонов ячеек, отведенных под неизвестные. Значения в этих ячейках должны изменяться в процессе поиска решения задачи так, чтобы найти решение, удовлетворяющее заданным ограничениям. В нашем случае введем в поле **Изменяя ячейки** диапазон `C10:G13`
Список **Ограничения**	Допускаются ограничения в виде равенств, неравенств, а также требований того, что неизвестные могут принимать только целые значения, либо только два значения: 0 или 1. Ограничения добавляются по одному за один раз и отображаются в окне **Добавление ограничения** (рис. 10.39), вызываемом нажатием кнопки **Добавить**. В поле **Ссылка на ячейку** введите левую часть ограничений — `C10:G13`, в поле **Ограничение** — правую часть, в нашем случае — 0. Раскрывающийся список позволяет задать тип соотношения между левой и правой частями ограничения. А именно ">=", "<=", "=", "цел" и "двоич". Соотношение "цел" подразумевает, что выражение может принимать только челочисленные значения, а "двоич" — только два значения: 0 и 1. В нашем случае выберите соотношение ">=". Таким образом, требование неотрицательности переменных задано. Нажмите кнопку **Добавить** и с помощью окна **Добавление ограничения** введите вторую группу ограничений налагаемых на переменные `C14:G14 = C15:G15`, а затем и третью `H10:G13 = I10:I13`. Нажмите кнопки **OK** для завершения ввода ограничений. На экране опять отобразится окно **Поиск решения**, но теперь уже заполненное

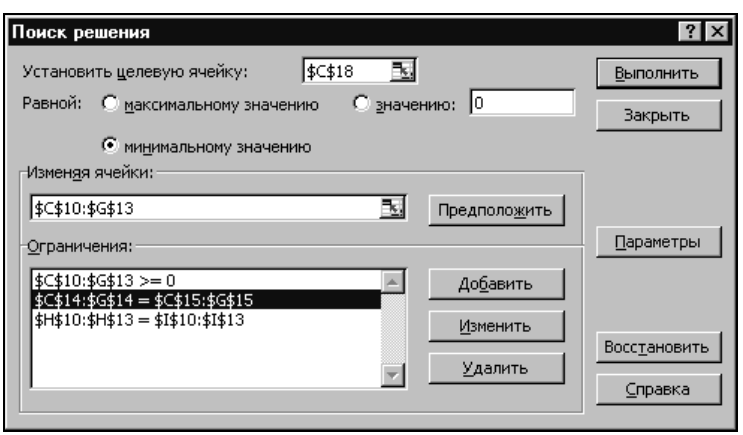


Рис. 10.38. Окно **Поиск решения** после заполнения для транспортной задачи

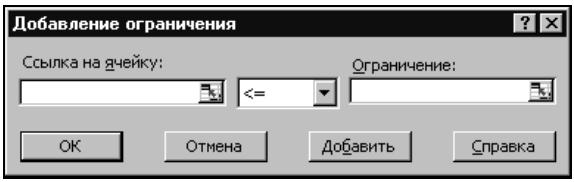


Рис. 10.39. Окно **Добавление ограничения**

Совет

Удобнее ограничения задавать в виде диапазонов, как это сделано в разобранном примере. Кроме того, обратите внимание, что при вводе ссылок на ячейки и диапазоны в окне **Поиск решения** они автоматически преобразуются в абсолютные ссылки.

2. Нажмите кнопку **Параметры**. На экране отобразится диалоговое окно **Параметры поиска решения** (рис. 10.40). В данном окне можно изменять условия и варианты поиска решения исследуемой задачи, а также загружать и сохранять оптимизируемые модели. Значения и состояния элементов управления, используемые по умолчанию, подходят для решения большинства задач. Элементы окна **Параметры поиска решения** описаны в табл. 10.17.

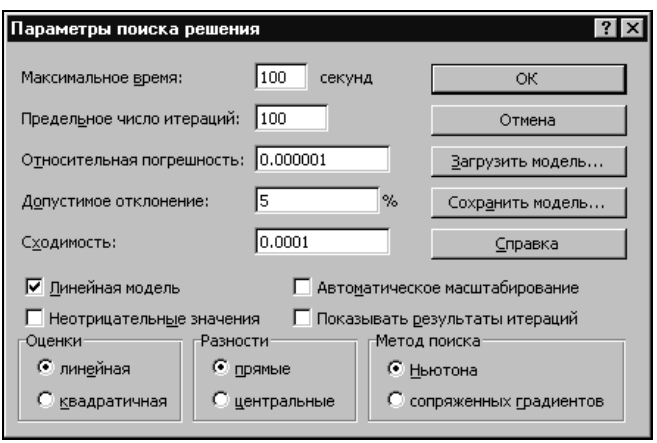


Рис. 10.40. Окно **Параметры поиска решения**

Таблица 10.17. *Элементы окна* ***Параметры поиска решения***

Элемент	Описание
Поле **Максимальное время**	Служит для ограничения времени, отпускаемого на поиск решения задачи

Таблица 10.17 *(окончание)*

Элемент	Описание
Поле **Предельное число итераций**	Служит для ограничения числа промежуточных вычислений
Поля **Относительная погрешность** и **Допустимое отклонение**	Служат для задания точности, с которой ищется решение. Рекомендуется после нахождения решения с величинами параметров, заданных по умолчанию, повторить вычисления с большей точностью и меньшим допустимым отклонением и сравнить с первоначальным методом. Использование подобной проверки особенно рекомендуется для задач с целочисленными ограничениями на переменные
Флажок **Линейная модель**	Служит для поиска решения линейной задачи оптимизации или линейной аппроксимации нелинейной задачи. Для нелинейной задачи этот флажок должен быть сброшен. Для линейной задачи — установлен, т. к. в противном случае возможно получение неверного результата
Флажок **Показывать результаты итераций**	Служит для приостановки поиска решения и просмотра результатов отдельных итераций
Флажок **Автоматическое масштабирование**	Служит для включения автоматической нормализации входных и выходных значений, качественно различающихся по величине. Например, максимизация прибыли в процентах по отношению к вложениям, исчисляемым в миллионах рублей
Группа **Оценка**	Служит для выбора метода экстраполяции
Группа **Разности**	Служит для выбора метода численного дифференцирования
Группа **Метод поиска**	Служит для выбора алгоритма оптимизации

В нашем конкретном случае установите флажок **Линейная модель**, а остальные значения, которые используются по умолчанию, можно оставить так, как есть. Нажмите кнопку **OK**. На экране опять отобразится окно **Поиск решения**.

3. Нажмите кнопку **Выполнить**. На экране отобразится окно **Результаты поиска решения** (рис. 10.41). После нажатия кнопки **OK** результаты будут внесены в рабочий лист (рис. 10.42). Средство Поиск решения нашло оптимальный план поставок продукции и соответствующие ему транспортные расходы.

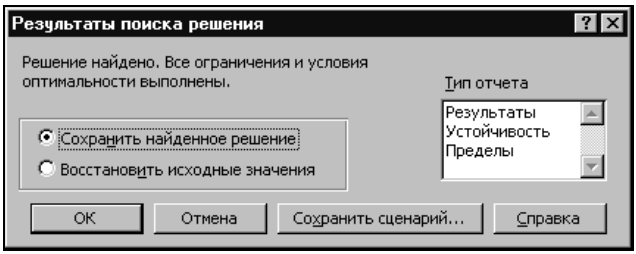


Рис. 10.41. Окно **Результаты поиска решения**

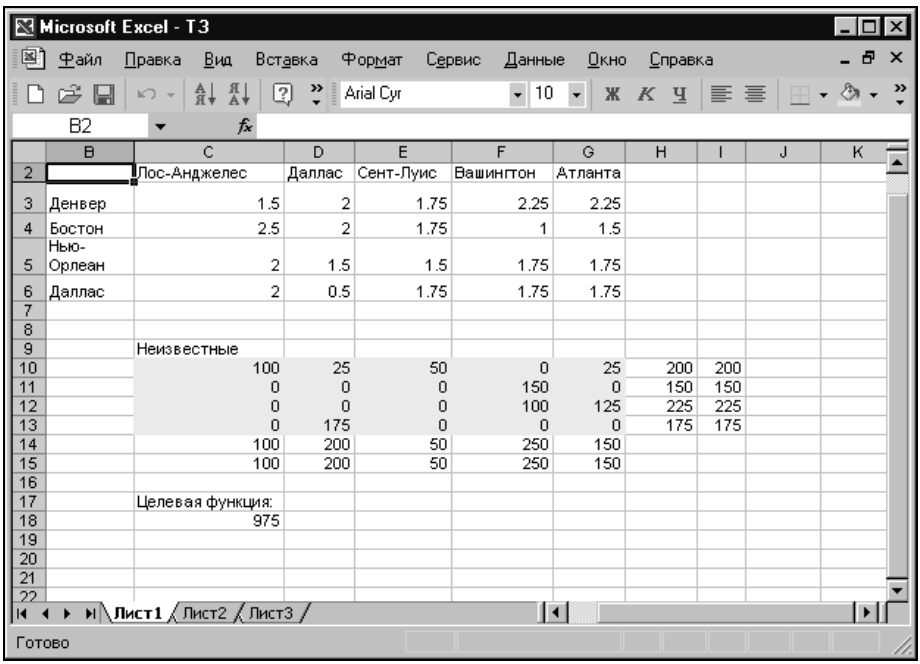


Рис. 10.42. Оптимальное решение транспортной задачи

Функции, программирующие поиск решения

После решения транспортной задачи вручную при помощи средства Поиск решения настало время обсудить, какие функции программируют процесс решения. Их не так много, а именно:

```
SolverAdd          SolverOk           SolverSave
SolverChange       SolverOptions      SolverSolve
SolverDelete       SolverReset        SolverSolve
SolverLoad
```

Функция `SolverOk` позволяет поставить задачу оптимизации. Она задает значения тех параметров, которые вручную устанавливаются в окне **Поиск решения**.

```
SolverOk(SetCell, MaxMinVal, ValueOf, ByChange)
```

- *SetCell* — задает ячейку, содержащую формулу с функцией цели;
- *MaxMinVal* — задает тип задачи, решаемой для функции цели. Допустимые значения:
  - 1 — для задачи максимизации;
  - 2 — для задачи минимизации;
  - 3 — для задачи нахождения значения;
- *ValueOf* — если значение параметра *MaxMinVal* равно 3, то этот параметр задает то значение, которое функция цели должна достичь;
- *ByChange* — задает диапазон изменяемых ячеек, т. е. тех ячеек, которые отведены под переменные оптимизационной задачи.

Функция `SolverAdd` позволяет добавлять в модель ограничения. Она задает значения тех параметров, которые вручную устанавливаются в окне **Добавление ограничения**.

```
SolverAdd(CellRef, Relation, FormulaText)
```

- *CellRef* — ссылка на ячейку или диапазон ячеек из левой части ограничений;
- *Relation* — целое число от 1 до 5, задающее тип соотношения между левой и правой частями ограничения. Если значение этого параметра равно 4 или 5, то значение параметр `FormulaText` опускается. Допустимые значения:
  - 1 — соответствует соотношению"<=";
  - 2 — соответствует соотношению"=";
  - 3 — соответствует соотношению">=";
  - 4 — допустимыми значениями диапазона ячеек, заданного параметром *CellRef*, являются целые числа;
  - 5 — допустимыми значениями диапазона ячеек, заданного параметром *CellRef*, являются только 0 и 1;
- *FormulaText* — ссылка на ячейку или диапазон ячеек из правой части ограничений либо значение из правой части ограничений.

Если список ограничений оптимизационной задачи уже задан, то в коде из него можно удалять ограничения функцией `SolverDelete`, а изменять огра-

ничения — функцией SolverChange. Синтаксис этих функций такой же, как у функции SolverAdd. А именно:

```
SolverDelete(CellRef, Relation, FormulaText)
SolverChange(CellRef, Relation, FormulaText)
```

Функция SolverOptions задает значения тех параметров поиска решения, которые вручную устанавливаются в окне **Параметры поиска решения**. Мы опустим описание этих параметров, т. к., по сути дела, они подробно рассмотрены при описании структуры окна **Параметры поиска решения**.

```
SolverOptions(MaxTime, Iterations, Precision, AssumeLinear, StepThru,
↳Estimates, Derivatives, Search, IntTolerance, Scaling, Convergence,
↳AssumeNonNeg)
```

Тесно с функцией SolverOptions связана функция SolverReset, которая устанавливает значения параметров поиска решения, используемые по умолчанию. У этой функции нет параметров.

Функция SolverSolve запускает поиск решения, ее действие эквивалентно нажатию кнопки **Выполнить** окна **Поиск решения**. Она возвращает значение 0, если решение найдено.

```
SolverSolve(UserFinish, ShowRef)
```

- *UserFinish* — параметр, принимающий логические значения. Если его значение равно False (которое устанавливается по умолчанию), результаты отображаются в диалоговом окне;
- *ShowRef* — ссылка на макрос, который должен выполняться между итерациями поиска решения.

Функция SolverSave сохраняет модель.

```
SolverSave(SaveArea)
```

Здесь параметр *SaveArea* задает диапазон ячеек, в котором сохраняется модель. Например:

```
SolverSave("Лист1!A1:A3")
```

Функция SolverLoad загружает модель.

```
SolverLoad(LoadArea)
```

Здесь параметр *LoadArea* задает диапазон ячеек, в котором записана модель.

Примечание

До использования перечисленных функций необходимо установить ссылку на Solver.xla в редакторе Visual Basic в окне **References**, отображаемом на экране выбором команды **Tools | References**. Если Solver.xla отсутствует в списке **Available References**, нажмите кнопку **Browse** и откройте Solver.xla из каталога C:\Program Files\Microsoft Office\Office10\Library.

Приложение Транспортная задача

Создадим простое приложение, решающее транспортную задачу. В этом приложении надо только указать ссылки на диапазоны ячеек, в которые введены стоимости перевозок и объемы ввоза продукции в пункты потребления и вывоза продукции из пунктов производства. Также необходимо привести ссылку на диапазон ячеек, которые отводятся под определяемые объемы перевозок. И это все, что надо задать пользователю. Все остальное сделает приложение, а именно введет в ячейки рабочего листа вспомогательные формулы, которые возвращают суммарную стоимость перевозок и объемы ввозимой и вывозимой продукции в каждом из пунктов. Приложение проверит модель на сбалансированность, задаст все параметры поиска решения и запустит его на выполнение. О результате нахождения решения приложение проинформирует пользователя соответствующей надписью на рабочем листе.

Перейдем к конструированию приложения. Создайте форму, на ней расположите четыре надписи, четыре элемента управления `RefEdit` и кнопку (рис. 10.43), и установите при помощи окна **Properties** значения их свойств, как показано в табл. 10.18.

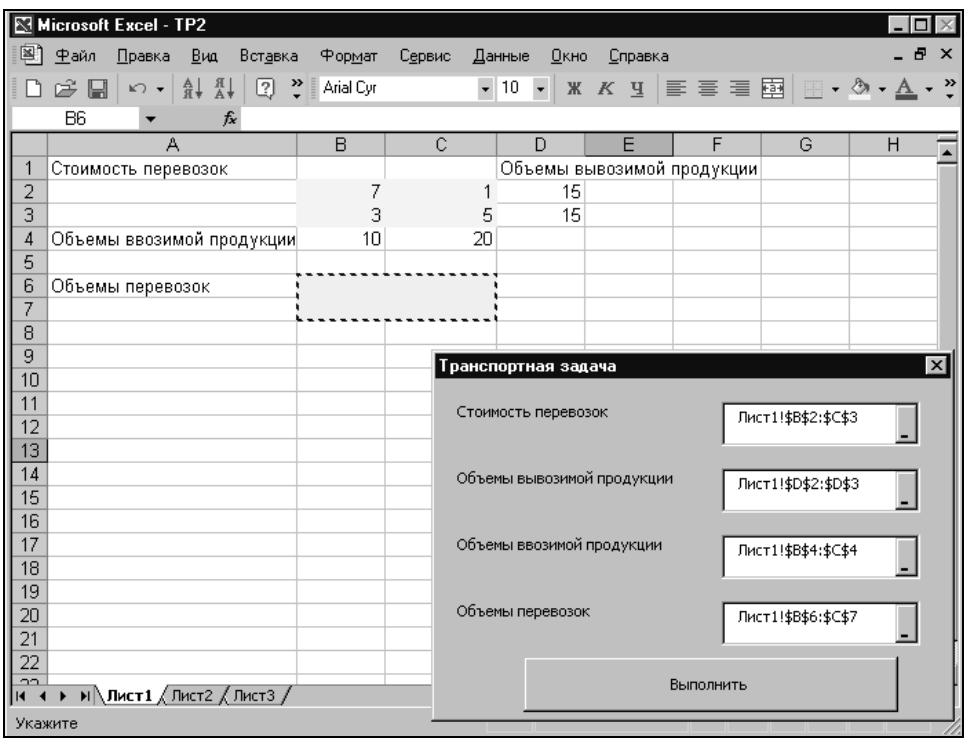


Рис. 10.43. Окно **Транспортная задача** и исходные данные на рабочем листе

Таблица 10.18. *Значения свойств, установленные в окне* ***Properties***

Объект	Свойство	Значение
Форма	Name	frmSolver
	Caption	Транспортная задача
Надпись	Caption	Стоимость перевозок
RefEdit	Name	refCosts
Надпись	Caption	Объемы вывозимой продукции
RefEdit	Name	refOut
Надпись	Caption	Объемы ввозимой продукции
RefEdit	Name	refIn
Надпись	Caption	Объемы перевозок
RefEdit	Name	refVar
Кнопка	Name	cmdSolve
	Caption	Выполнить

Прежде чем набирать код (листинг 10.12), убедитесь, что установлена ссылка на Solver.xla в окне **References**, отображаемом на экране выбором команды **Tools | References**. Если Solver.xla отсутствует в списке **Available References**, нажмите кнопку **Browse** и откройте Solver.xla из каталога C:\Program Files\Microsoft Office\Office10\Library.

На рис. 10.44 приведены результаты выполнения приложения Транспортная задача (как введенные вспомогательные формулы в ячейки рабочего листа, так и результат расчета).

Листинг 10.12. Решение транспортной задачи

```
Option Explicit

Private Sub cmdSolve_Click()
   Dim vars As String, Costs As String
   Dim Target As String
   Dim sOut As String, sIn As String
   Dim varIn As String, varOut As String
   Dim r As Integer, c As Integer
   Dim vIn As Double, vOut As Double
   Dim res As Integer

   vars = refVar.Text
   sIn = refIn.Text
```

```
   sOut = refOut.Text
   Costs = refCosts.Text
' Проверка сбалансировки модели
   vIn = Evaluate("SUM(" & sIn & ")")
   vOut = Evaluate("SUM(" & sOut & ")")
   If vIn <> vOut Then
      MsgBox "Задача не сбалансирована", _
              vbExclamation, _
             "Транспортная задача"

      Exit Sub
   End If

   With Range(vars)
      r = .Rows.Count
      c = .Columns.Count
      Target = .Cells(r + 1, c + 1).Address(RowAbsolute:=True, _
                                            ColumnAbsolute:=True)
   End With

   Range(Target).FormulaLocal = "=СУММПРОИЗВ(" & _
                                vars & ";" & Costs & ")"

   With Range(vars)
      varIn = Range(.Cells(r + 1, 1), _
                    .Cells(r + 1, c)).Address(RowAbsolute:=True, _
                                              ColumnAbsolute:=True)
      varOut = Range(.Cells(1, c + 1), _
                     .Cells(r, c + 1)).Address(RowAbsolute:=True, _
                                               ColumnAbsolute:=True)
   End With
   TranAbsRef varIn, varIn
   TranAbsRef varOut, varOut

'Ввод вспомогательных формул в ячейки рабочего листа
   Range(vars).Cells(r + 1, 1).Select
   ActiveCell.FormulaR1C1 = "=SUM(R[-" & CStr(r) & "]C:R[-1]C)"
   Selection.AutoFill Destination:=Range(varIn), _
                      Type:=xlFillDefault

   Range(vars).Cells(1, c + 1).Select
   ActiveCell.FormulaR1C1 = "=SUM(RC[-" & CStr(c) & "]:RC[-1])"
   Selection.AutoFill Destination:=Range(varOut), _
                      Type:=xlFillDefault
```

```
' Поиск решения
   SolverReset
   SolverOk SetCell:=Target, _
            MaxMinVal:=2, _
            ValueOf:="0", _
            ByChange:=vars
            SolverAdd CellRef:=vars, _
            Relation:=3, _
            FormulaText:="0"
   SolverAdd CellRef:=varIn, _
             Relation:=2, _
             FormulaText:=sIn
   SolverAdd CellRef:=varOut, _
             Relation:=2, _
             FormulaText:=sOut
   SolverOptions MaxTime:=100, Iterations:=100, Precision:=0.000001, _
                 AssumeLinear:=True, StepThru:=False, Estimates:=1, _
                 Derivatives:=1, SearchOption:=1, IntTolerance:=5, _
                 Scaling:=False, Convergence:=0.0001, AssumeNonNeg:=False
   res = SolverSolve(UserFinish:=True)
   With Range(vars).Cells(r + 2, c + 2)
      If res = 0 Then
         .Value = "Решение найдено"
      Else
         .Value = "Решение не найдено"
      End If
   End With
End Sub
```

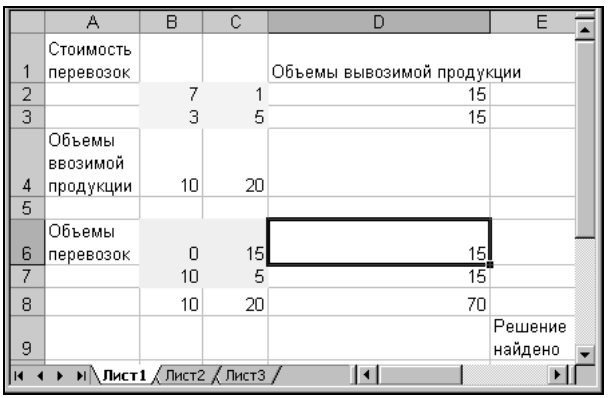
Примечание

Обратите внимание, как в программе использован метод `Evaluate` для преобразования выражения в значение. Этот трюк избавил нас от необходимости либо добавить в код два цикла для определения суммарных объемов ввозимой и вывозимой продукции, либо вводить в ячейки рабочего листа ненужные дополнительные формулы.

Примечание

Использование в программе функции `SolverReset` до задания параметров поиска решения существенно. Устанавливая значения параметров поиска решения равными тем значениям, которые используются по умолчанию, функция `SolverReset` тем самым очищает окно **Поиск решения** от всех предыдущих установок. Если не использовать функцию `SolverReset`, то это приведет к добавлению к списку тех ограничений, которые задаются функциями `SolverAdd`

при каждом нажатии кнопки **Выполнить**. Легко вообразить, что в этом случае произойдет со списком **Ограничения** окна **Поиск решения**, например, через десяток нажатий кнопки **Выполнить**!



	A	B	C	D	E
1	Стоимость перевозок			Объемы вывозимой продукции	
2		7	1	15	
3		3	5	15	
4	Объемы ввозимой продукции	10	20		
5					
6	Объемы перевозок	0	15	15	
7		10	5	15	
8		10	20	70	
9					Решение найдено

Лист1 Лист2 Лист3

	A	B	C	D	E
1	Стоимость перевозок			Объемы вывозимой продукции	
2		7	1	15	
3		3	5	15	
4	Объемы ввозимой продукции	10	20		
5					
6	Объемы перевозок	0	15	=СУММ(B6:C6)	
7		10	5	=СУММ(B7:C7)	
8		=СУММ(B6:B7)	=СУММ(C6:C7)	=СУММПРОИЗВ(Лист1!$B$6:$C$7;Лист1!$B$2:$C$3)	
9					Решение найдено

Лист1 Лист2 Лист3

Рис. 10.44. Результат расчета приложения Транспортная задача

Решение оптимизационных задач, зависящих от параметра

Программирование поиска решения может сослужить хорошую службу и существенно ускорить процесс обработки данных, когда требуется проанализировать зависимость оптимального решения от параметра. Покажем это на примере простейшей задачи. Предположим, что нам требуется найти решение системы нелинейных уравнений:

$$x^2 + y^2 - 1 = 0,$$
$$2x + 3y - d = 0,$$

где параметр d изменяется в интервале от 0,2 до 1 с шагом 0,1. Очевидно, что решить данную систему равносильно нахождению решения уравнения:

$$(x^2 + y^2 - 1)^2 + (2x + 3y - d)^2 = 0.$$

Первоначальная система уравнений, конечно, не решается средством Поиск решения, а вот равносильное ей уравнение с двумя неизвестными подходит для того, чтобы попытаться его решить.

Итак, на рабочем листе отведите ячейки **A1** и **B1** под неизвестные, ячейку **D1** под значения параметра, а в ячейку **C1** введите формулу левой части уравнения

```
=(A1^2+B1^2-1)^2+(2*A1+3*B1-D1)^2
```

На рабочем листе все готово для применения поиска решения. Перейдем к конструированию интерфейса приложения. Создайте форму с тремя надписями, тремя полями ввода и кнопкой (рис. 10.45). Установите значения свойств этих элементов управления с помощью окна **Properties**, как показано в табл. 10.19.

Таблица 10.19. *Значения свойств, установленные в окне* ***Properties***

Объект	Свойство	Значение
Форма	Name	frmSystemSolver
	Caption	Решение системы, зависящей от параметра
Надпись	Caption	Начальное значение
Поле	Name	txtBegin
Надпись	Caption	Конечное значение
Поле	Name	txtEnd
Надпись	Caption	Шаг изменения
Поле	Name	txtStep
Кнопка	Name	cmdOK
	Caption	OK

Прежде чем набирать код (листинг 10.13), убедитесь, что установлена ссылка на Solver.xla в окне **References**, отображаемом на экране выбором команды **Tools | References**. Если Solver.xla отсутствует в списке **Available References**, нажмите кнопку **Browse** и откройте Solver.xla из каталога C:\Program Files\Microsoft Office\Office10\Library.

Итак, для решения задачи достаточно в поля ввода ввести начальное, конечное значения параметра и шаг его изменения. После нажатия кнопки **OK** приложение найдет решения и выведет в столбец **E** значения параметра, в столбцы **F** и **G** значения неизвестных *x* и *y*.

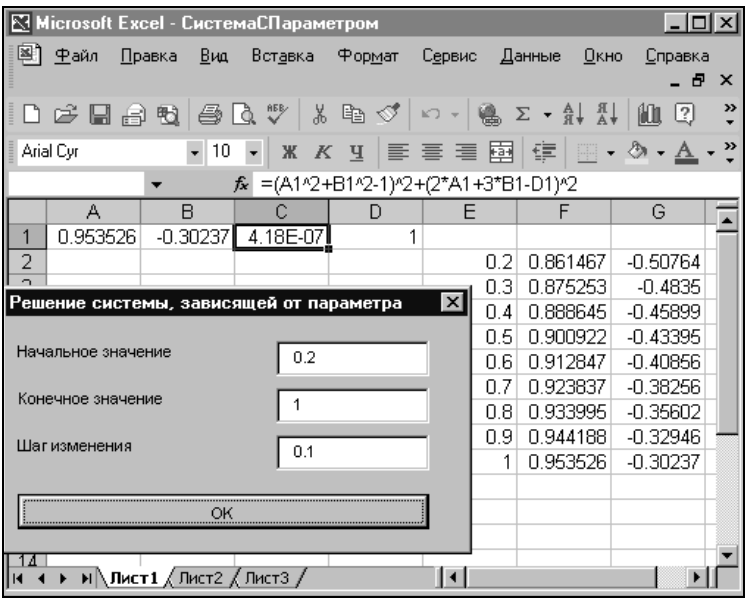


Рис. 10.45. Окно **Решение системы, зависящей от параметра**

Листинг 10.13. Решение системы уравнений, зависящих от параметра

```
Private Sub cmdOK_Click()
   Dim i As Integer, flag As Integer
   Dim a As Double, b As Double, h As Double, d As Double
   a = CDbl(txtBegin.Text)
   b = CDbl(txtEnd.Text)
   h = CDbl(txtStep.Text)
   d = a
   i = 2
   Do
      Range("A1").Value = 0
      Range("B1").Value = 0
      Range("D1").Value = d
      SolverReset
      SolverOk SetCell:="$C$1", MaxMinVal:=3, _
               ValueOf:="0", ByChange:="$A$1:$B$1"
```

```
      SolverOptions MaxTime:=100, Iterations:=100, _
                    Precision:=0.000001, AssumeLinear:=False, _
                    StepThru:=False, Estimates:=1, Derivatives:=1, _
                    SearchOption:=1, IntTolerance:=5, Scaling:=False, _
                    Convergence:=0.0001, AssumeNonNeg:=False
      SolverOk SetCell:="$C$1", MaxMinVal:=3, _
               ValueOf:="0", ByChange:="$A$1:$B$1"

   flag = SolverSolve(UserFinish:=True)

   Cells(i, 5).Value = d
   Cells(i, 6).Value = Range("A1").Value
   Cells(i, 7).Value = Range("B1").Value
   If flag <> 0 Then Cells(i, 8).Value = "Решение не найдено"
      i = i + 1
      d = d + h
   Loop Until d > b
End Sub
```

Примечание

С геометрической точки зрения рассмотренная в данном разделе задача представляет собой нахождение точки пересечения прямой и окружности. Понятно, что у этой задачи либо нет решения (прямая не пересекается с окружностью), либо имеется только одно (прямая касается окружности), либо два решения (прямая пересекает окружность). Наша программа нашла только половину решений. Это связано с тем, что все решения мы находили с помощью одного и того же начального приближения, а именно нулевого. В качестве упражнения модернизируйте приложение так, чтобы оно находило все решения задачи.

Решение уравнения, зависящего от двух параметров

Рассмотрим простую задачу определения точки пересечения эллипса с биссектрисой первого квадранта, т. е. решение уравнения:

$$x^2/a^2 + y^2/b^2 - 1 = 0$$

при $x = y$.

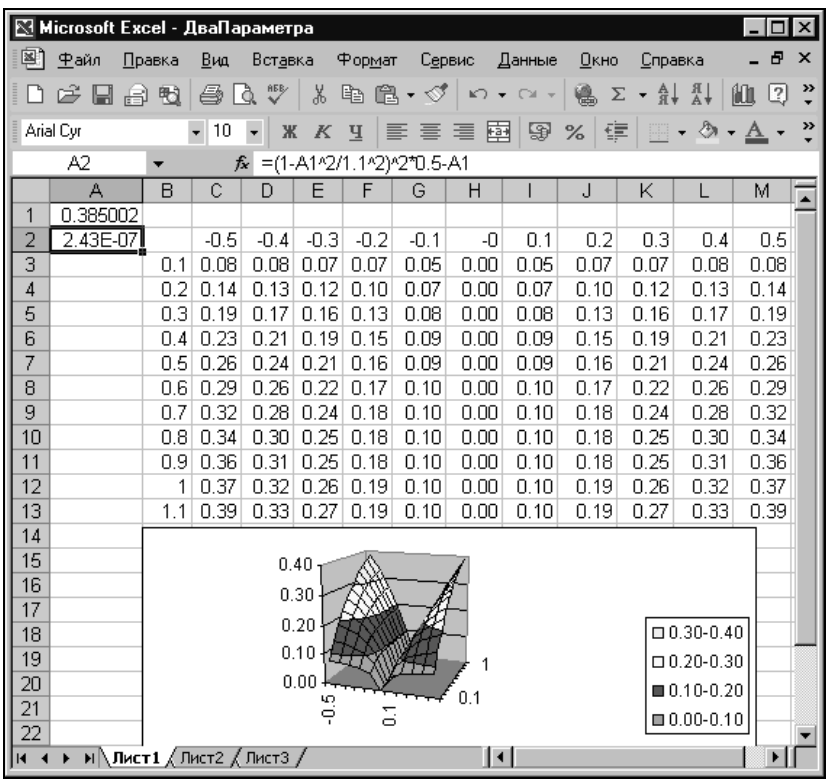
Очевидно, что абсцисса точки пересечения определяется как решение уравнения:

$$|b|\,(1 - x^2 / a^2)^{1/2} - x = 0.$$

Найдем неотрицательные решения уравнения при значениях параметров a и b, изменяемых соответственно от 0,1 до 1,1 и от $-0{,}5$ до 0,5 с одним и тем же шагом 0,1. Для решения этой задачи подходит средство Поиск решения. Если бы не было ограничения на неотрицательность корня, то можно было

бы ограничиться подбором параметра, т. к. в задаче только одно неизвестное, а подбор параметра как раз и предназначен для решения уравнений с одним неизвестным.

При решении данного примера мы не будем создавать удобный интерфейс для ввода данных, оставляя это сделать читателю, а ограничимся только демонстрацией кода, непосредственно решающего данную задачу (листинг 10.14). Особенностью применения поиска решения в данном случае является то, что, несмотря на многократное использование поиска решения, сами параметры поиска решения в коде задаются только один раз. Затем поиск решения просто многократно выполняется при различных значениях параметров *a* и *b*, а из найденных решений составляется таблица (рис. 10.46).



	A	B	C	D	E	F	G	H	I	J	K	L	M
1	0.385002												
2	2.43E-07		-0.5	-0.4	-0.3	-0.2	-0.1	-0	0.1	0.2	0.3	0.4	0.5
3		0.1	0.08	0.08	0.07	0.07	0.05	0.00	0.05	0.07	0.07	0.08	0.08
4		0.2	0.14	0.13	0.12	0.10	0.07	0.00	0.07	0.10	0.12	0.13	0.14
5		0.3	0.19	0.17	0.16	0.13	0.08	0.00	0.08	0.13	0.16	0.17	0.19
6		0.4	0.23	0.21	0.19	0.15	0.09	0.00	0.09	0.15	0.19	0.21	0.23
7		0.5	0.26	0.24	0.21	0.16	0.09	0.00	0.09	0.16	0.21	0.24	0.26
8		0.6	0.29	0.26	0.22	0.17	0.10	0.00	0.10	0.17	0.22	0.26	0.29
9		0.7	0.32	0.28	0.24	0.18	0.10	0.00	0.10	0.18	0.24	0.28	0.32
10		0.8	0.34	0.30	0.25	0.18	0.10	0.00	0.10	0.18	0.25	0.30	0.34
11		0.9	0.36	0.31	0.25	0.18	0.10	0.00	0.10	0.18	0.25	0.31	0.36
12		1	0.37	0.32	0.26	0.19	0.10	0.00	0.10	0.19	0.26	0.32	0.37
13		1.1	0.39	0.33	0.27	0.19	0.10	0.00	0.10	0.19	0.27	0.33	0.39

Рис. 10.46. Решение уравнения, зависящего от двух параметров

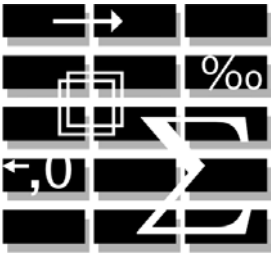
Листинг 10.14. Решение уравнения с двумя параметрами

```
Sub TwoParameters()
   Dim Ab As Double, Ae As Double, Ast As Double
   Dim Bb As Double, Be As Double, Bst As Double
```

```
   Dim a As Double, b As Double
   Dim i As Integer, j As Integer
   Dim Flag As Integer

' Начальное, конечное значения параметра a и шаг его изменения
   Ab = 0.1: Ae = 1.1: Ast = 0.1
' Начальное, конечное значения параметра b и шаг его изменения
   Bb = -0.5: Be = 0.5: Bst = 0.1
   a = Ab
   b = Bb
' Задание параметров поиска решения:
' Уравнение вводится в ячейку A2, а под неизвестное отводится ячейка A1
   SolverReset
   Range("A2").Value = "=(1-A1^2/" & CStr(a) & ")^2*" & CStr(b) & "-A1"
   SolverOk SetCell:="$A$2", MaxMinVal:=3, _
            ValueOf:="0", ByChange:="$A$1"
   SolverAdd CellRef:="$A$1", Relation:=3, FormulaText:="0"
' Последовательное решение при различных комбинациях значения параметра
' уравнения. Найденное решение считывается из ячейки A1 и последовательно
' вводится в ячейки диапазона с верхним левым углом в ячейке C3.
' Во вторую строку выводятся значения параметра b.
' Во второй столбец выводятся значения параметра a.
   i = 3
   Do
      b = Bb
      j = 3
      Do
      DoEvents
         Range("A1").Value = a
         Range("A2").Value = "=(1-A1^2/" & CStr(a) & "^2)^2*" & _
                             CStr(Abs(b)) & "-A1"
         Flag = SolverSolve(UserFinish:=True)
         If Flag = 0 Then
            Cells(i, j).Value = Range("A1").Value
         Else
            Cells(i, j).Value = Empty
         End If
            Cells(2, j).Value = b
         j = j + 1
         b = b + Bst
      Loop Until b > Be
      Cells(i, 2).Value = a
      i = i + 1
      a = a + Ast
   Loop Until a > Ae
End Sub
```

ГЛАВА 11



Обработка ошибок и отладка программ

При составлении приложений важно предусмотреть, чтобы программа анализировала возможные ошибки, возникающие при ее выполнении по вине пользователя, и информировала его об этом, подсказывая пользователю, что конкретно он сделал неправильно. При этом возможно два подхода:

- *предотвращающий ошибки.* Программно анализируются вводимые или вычисляемые данные и в случае, если они могут приводить к ошибке, обеспечивается, чтобы программа информировала пользователя о необходимости корректного задания данных;
- *обрабатывающий ошибки.* В случае появления ошибки, она перехватывается, обрабатывается и создается программный отклик на возникшую ошибку.

При создании приложений надо сочетать оба подхода, применяя в каждом конкретном случае и для каждой возможной ошибки тот подход, который кажется разработчику наиболее эффективным.

Разработка процедур, предотвращающих появление ошибок

Рассмотрим процесс создания приложения, в котором предотвращается появление ошибок на простейшем примере. В диалоговом окне имеется только два поля ввода: **Исходное значение** — для ввода числа, **Обратное значение** — для отображения в нем обратной величины к введенному значению (рис. 11.1). По нажатии кнопки **OK** находится обратное значение и выводится в поле ввода **Обратное значение**.

Для реализации приложения создайте форму с двумя надписями, двумя полями ввода и кнопкой. Установите значения свойств элементов управления, как показано в табл. 11.1.

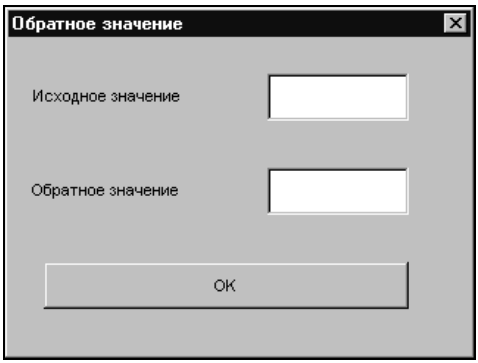


Рис. 11.1. Диалоговое окно **Обратное значение**

Таблица 11.1. *Значения свойств, установленные в окне* ***Properties***

Элемент управления	Свойство	Значение
Надпись	Caption	Исходное значение
Поле ввода	Name	txtNumber
Надпись	Caption	Обратное значение
Поле ввода	Name	txtInverse
Кнопка	Name	cmdOK
	Caption	OK

Следующая программа (листинг 11.1) находит обратное значение без контроля за появлением возможных ошибок.

Листинг 11.1. Нахождение обратного значения без предупреждения о возможных ошибках

```
Private Sub cmdOK_Click()
   Dim x As Double, y As Double
   x = CDbl(txtNumber.Text)
   y = 1 / x
   txtInverse.Text = CStr(y)
End Sub
```

Несмотря на то, что рассматриваемая ситуация очень простая, уже она таит в себе множество подводных камней. Например, если пользователь по невнимательности забудет ввести в поле **Исходное значение** число, при нажатии кнопки **OK** произойдет аварийное прерывание выполнения программы

с мало понятным сообщением о несоответствии типов, отображаемым в диалоговом окне **Microsoft Visual Basic** (рис. 11.2).

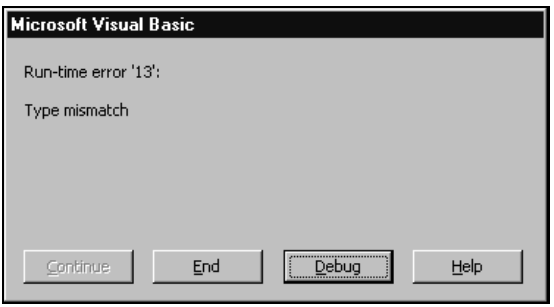


Рис. 11.2. Окно **Microsoft Visual Basic**

Данное сообщение об ошибке связано с инструкцией:

```
x = CDbl(txtNumber.Text)
```

где параметром функции `CDbl` должна быть строка, преобразуемая в число. Если в поле **Исходное значение** не введено ничего, по умолчанию из этого поля будет считываться пустая строка. Но пустая строка не может быть преобразована в число, и поэтому из-за функции `CDbl` происходит ошибка. Ошибка из-за несоответствия типов возникнет также, если в одно из полей пользователь по неосторожности введет число с десятичной запятой, а установками системы предусматривается десятичная точка, и наоборот.

Данные ошибки ввода легко избежать, предусмотрев в программе предварительную проверку: преобразуются ли вводимые данные в числа? Эту предварительную проверку можно сделать, например, как показано в процедуре (листинг 11.2). Кроме того, в этой процедуре заодно проверяется, не является ли исходное значение нулем. Деление на ноль тоже может вызвать ошибку. О возникшей ошибке программа информирует пользователя, устанавливает фокус на поле ввода **Исходное значение** и ожидает от пользователя корректного ввода числа.

Листинг 11.2. Нахождение обратного значения. Деление чисел с проверкой корректности вводимых данных

```
Private Sub cmdOK_Click()
    Dim x As Double, y As Double
    If Not IsNumeric(txtNumber.Text) Then
        MsgBox "Некорректный ввод исходного значения", _
               vbInformation, "Обратное значение"
        txtNumber.SetFocus
        Exit Sub
    End If
```

```
    x = CDbl(txtNumber.Text)
    If x = 0 Then
        MsgBox "Введен ноль", vbInformation, "Обратное значение"
        txtNumber.SetFocus
        Exit Sub
    End If
    y = 1 / x
    txtInverse.Text = CStr(y)
End Sub
```

Контроль вводимых значений с помощью обработки события *KeyPress*

В примере из предыдущего раздела данные в поле ввода сначала вводились, а потом проводилась проверка корректности их ввода. Можно поступить иначе, проводя контроль непосредственно на этапе ввода и отсеивая неправильно вводимые значения. Этого можно достичь, добавив в программу код обработки события `KeyPress` поля ввода. Например, если согласно постановке задачи исходным числом может быть только целое неотрицательное число, то в код программы можно добавить следующую процедуру (листинг 11.3), которая обеспечивает ввод только цифр.

Листинг 11.3. Обеспечение ввода только целых положительных чисел

```
Private Sub txtNumber_KeyPress(ByVal KeyAscii As MSForms.ReturnInteger)
   Dim ЦелоеЧисло As String
   ЦелоеЧисло = "0123456789"
   If KeyAscii > 26 Then
      If InStr(ЦелоеЧисло, Chr(KeyAscii)) = 0 Then
         KeyAscii = 0
      End If
   End If
End Sub
```

Примечание

В процедуре в первую очередь проверяется, был ли код нажатой клавиши больше 26. Это наибольший управляющий код VBA. Управляющий код указывает на то, что нажата не обычная клавиша клавиатуры, а управляющая, такая как <Esc>, <Del>, <Backspace> и т. д. После этого процедура проверяет, была ли нажата одна из цифровых клавиш. Если не была нажата цифровая клавиша, то параметру `KeyAscii` присваивается нулевое значение, что приводит к игнорированию нажатия клавиши. Таким образом, данный код обеспечивает ввод с клавиатуры в поле ввода только цифр и редактирование введенного числа стандартными средствами при помощи управляющих клавиш.

Примечание

При обработке события KeyPress можно не только контролировать, что вводится в поле ввода, но и автоматически делать исправления вводимой информации. Например, в следующем коде (листинг 11.4) все вводимые буквы автоматически становятся прописными.

Листинг 11.4. Автоматический перевод вводимых букв в прописные

```
Private Sub txtInput_KeyPress(ByVal KeyAscii As MSForms.ReturnInteger)
   KeyAscii = Asc(UCase(Chr(KeyAscii)))
End Sub
```

Давайте немного усложним ситуацию. Пусть исходным числом может быть любое число, записанное в формате числа с фиксированной десятичной точкой. В этом случае надо проверить, не является ли набираемый символ либо цифрой, либо знаком, либо точкой:

- если не является, то набор символа игнорируется;
- если набираемый символ является цифрой, то он вводится в поле ввода;
- если набираемый символ является точкой, то проверяется, является ли она первой вводимой точкой. Если ранее точки не вводились, она вводится в поле ввода. В противном случае ввод точки игнорируется;
- если набираемый символ является знаком, то проверяется, в какой позиции числа он вводится. Если в первой позиции, то знак вводится. В противном случае ввод знака блокируется.

Данный подход реализован в следующем коде (листинг 11.5).

Листинг 11.5. Обеспечение ввода только действительных чисел

```
Private Sub txtNumber_KeyPress(ByVal KeyAscii As MSForms.ReturnInteger)
   Dim Число As String
   Dim Знак As String
   Число = "0123456789.+-"
   Знак = "+-"
   If KeyAscii > 26 Then
      If InStr(Число, Chr(KeyAscii)) = 0 Or _
        (InStr(txtNumber.Text, ".") > 0 And Chr(KeyAscii) = ".") Or _
        (txtNumber.SelStart > 0 And InStr(Знак, Chr(KeyAscii)) > 0) Then
         KeyAscii = 0
      End If
   End If
End Sub
```

Перехват и обработка ошибок

Из диалогового окна **Microsoft Visual Basic** (см. рис. 11.2) видно, что каждая ошибка имеет свой код. В табл. 11.2 приведены коды наиболее часто встречаемых ошибок.

Таблица 11.2. *Коды наиболее часто встречаемых ошибок*

Код	Сообщение
5	Приложение не запущено
6	Переполнение
7	Не хватает памяти
9	Индекс выходит за пределы допустимого диапазона
11	Деление на ноль
13	Несоответствие типа
18	Произошло прерывание, вызванное пользователем
52	Неправильное имя файла или идентификатора
53	Файл не найден
54	Неверный режим работы с файлом
55	Файл уже открыт
56	Ошибка ввода/вывода
61	Переполнение диска
68	Устройство недоступно
71	Диск не готов
72	Повреждена поверхность диска
335	Невозможен доступ к системным ресурсам
368	Истек срок действия данного файла. Программе требуется файл более новой версии
482	Ошибка принтера

В идеале разрабатываемое приложение не должно никогда аварийно прерываться. В приложении должны быть созданы средства перехвата любой возможной для этого приложения ошибки, обработки ее, выдачи сообщения

пользователю и безаварийное продолжение работы приложения. Обычно конструкция перехвата ошибок имеет следующую структуру:

```
Оператор On Error
Процедура обработки ошибки
Оператор Resume
```

Оператор *On Error*

Оператор `On Error` производит перехват ошибки. Он устанавливает, что программа должна делать в случае появления ошибки. Допустимы следующие варианты синтаксиса:

- `On Error GoTo` *`line`*

  Активизирует процедуру обработки ошибок, начало которой определяется обязательным параметром *`line`*. Его значением может быть либо метка строки, либо номер строки.

- `On Error Resume Next`

  Указывает, что при возникновении ошибки происходит передача управления на инструкцию, непосредственно следующую за инструкцией, где возникла ошибка.

- `On Error GoTo 0`

  Отключает любой активизированный обработчик ошибок в текущей процедуре.

Процедура обработки ошибки

Процедура обработки ошибки определяет тип возникшей ошибки и устанавливает, что программа должна делать в зависимости от типа ошибки.

Оператор *Resume*

Обеспечивает процедуре возможность продолжить работу после обработки ошибки. Допустимы три варианта синтаксиса этого оператора.

- `Resume`

  После обработки ошибки управление передается той инструкции, в которой произошла ошибка.

- `Resume` *`line`*

  После обработки ошибки управление передается инструкции, определенной параметром *`line`*. Значением этого параметра может быть любая метка строки или номер строки.

❒ `Resume Next`

После обработки ошибки управление передается инструкции, следующей за инструкцией, в которой произошла ошибка.

Объект *Err*

Процедура обработки ошибки обычно включает объект `Err`, который инкапсулирует в себе всю информацию о последней ошибке, возникшей во время выполнения приложения. В табл. 11.3 и 11.4 перечислены свойства и методы этого объекта.

Таблица 11.3. *Свойства объекта* `Err`

Свойство	Описание
`Number`	Возвращает код ошибки
`Source`	Имя программного файла, в котором возникла ошибка
`Description`	Возвращает строковое выражение, содержащее текст сообщения об ошибке, соответствующий коду ошибки
`HelpFile`	Полное имя (включая диск и путь) файла справки. Этот файл может быть вызван для поддержки сообщения об ошибке
`HelpContext`	Идентификатор раздела в справочном файле, указанном в свойстве `HelpFile`
`LastDLLError`	Содержит системный код ошибки для последнего вызова библиотеки динамической компоновки (DLL)

Таблица 11.4. *Методы объекта* `Err`

Метод	Описание
`Clear`	Очищает все значения свойств объекта `Err`. Метод `Clear` используется для явной очистки значений свойств объекта `Err` после завершения обработки ошибки. Это необходимо, например, при отложенной обработке ошибки, которая задается оператором `On Error Resume Next`
`Raise`	Создает ошибку выполнения. Используется при моделировании ситуаций ошибки. `Raise` *`number`*, *`source`*, *`description`*, *`helpfile`*, *`helpcontext`* ♦ *`number`* — номер ошибки, т. е. целое число от 0 до 65535; ♦ *`source`* — строковое выражение, определяющее имя объекта или приложения, в котором возникла ошибка;

Таблица 11.4 (окончание)

Метод	Описание
(прод.)	♦ *description* — строковое выражение, содержащее описание ошибки; ♦ *helpfile* — полное имя (включая диск и путь) файла справки Microsoft Windows, содержащего описание данной ошибки; ♦ *helpcontext* — контекстный идентификатор, определяющий раздел в файле, указанном в параметре Раздел, соответствующий обрабатываемой ошибке

Принципиальным отличием применения объекта Err от подхода, использованного в предыдущем разделе, является то, что объект Err позволяет реагировать на возникшую в приложении ошибку, а не создавать средства, обеспечивающие проверку корректности данных до начала работы с ними. В этом смысле с помощью объекта Err возможно конструирование приложений, более гибко реагирующих на возникающие в них ошибки, причем это происходит за счет сокращения и упрощения кода, что также является немаловажным достоинством объекта Err.

На конкретном примере покажем, как применяется объект Err при создании обработчика ошибок. В *разд. "Разработка процедур, предотвращающих появление ошибок"* в процессе создания диалогового окна **Обратное значение** и связанной с ним программы, на первый взгляд, были предусмотрены все возможные ошибки. Но это только на первый взгляд. Введем, например, в поле **Исходное значение** величину 1E-310. Это число ничем не лучше или не хуже любого другого числа типа Double. Тем не менее вместо вывода результата произойдет аварийное прерывание выполнения программы с отображением сообщения об ошибке переполнения. Усовершенствуем нашу программу, предусмотрев возможность обработки как ранее описанных ошибок, так и ошибки переполнения (листинг 11.6). В обработчике ошибок предусмотрим следующие возможности отклика:

- пользователь информируется об ошибке переполнения (ошибка 16) при вводе данных. В этом случае как некорректно введенное данное, так и результат по умолчанию полагаются равным единице. Процедура обработки события Click кнопки **OK** продолжает работу с инструкцией, помеченной меткой ReturnPoint;
- пользователь информируется об ошибке деления на ноль (ошибка 11). В этом случае поле ввода числа получает фокус, а процедура обработки события Click кнопки завершает работу;
- пользователь информируется об ошибке несоответствия типов (ошибка 13). В этом случае пользователя просят ввести число в поле ввода, которое получает фокус, а процедура обработки события Click кнопки завершает работу;

- если при выполнении программы возникнут какие-то непредвиденные ошибки, то о них информируется пользователь и прерывается выполнение процедуры обработки события `Click` кнопки;
- в обработчике ошибок имеется ветвь `Case Else` для обработки непредвиденных ошибок.

Листинг 11.6. Обработчик ошибок

```
Private Sub cmdOK_Click()
   Dim x As Double, y As Double
   On Error GoTo ErrorHandler
   x = CDbl(txtNumber.Text)
   y = 1 / x
ReturnPoint:
   txtInverse.Text = CStr(y)
Exit Sub

ErrorHandler:
   MsgBox "Произошла ошибка: " & CStr(Err.Number) & "-" & _
                           Err.Description, vbInformation, _
                          "Обратное значение"
   Select Case Err.Number
      Case 6
         MsgBox "Переполнения. " & vbCr & _
                "Установлены значения по умолчанию."
         txtNumber.Text = 1
         y = 1
         txtNumber.SetFocus
         Resume ReturnPoint
      Case 11
         MsgBox "На ноль делить нельзя"
         txtNumber.SetFocus
         Exit Sub
      Case 13
         MsgBox "Должно быть число"
         txtNumber.SetFocus
         Exit Sub
      Case Else
         MsgBox "Непредвиденная ошибка"
         Exit Sub
   End Select
End Sub
```

```
Private Sub cmdOK_Click()
   Dim x As Double, y As Double
   On Error GoTo ErrorHandler
      x = CDbl(txtNumber.Text)
      y = 1 / x
ReturnPoint:
      txtInverse.Text = CStr(y)
Exit Sub

ErrorHandler:
   MsgBox "Произошла ошибка: " & CStr(Err.Number) & "-" & _
                           Err.Description, vbInformation, _
                          "Обратное значение"
   Select Case Err.Number
    Case 6
        txtNumber.Text = 1
        y = 1
        txtNumber.SetFocus
        Resume ReturnPoint
    Case 11
        txtNumber.SetFocus
        Exit Sub
    Case 11
        txtNumber.Text = 1
        Resume
    Case Else
        MsgBox "Непредвиденная ошибка"
        Exit Sub
   End Select
End Sub
```

Примечание

В случае, если разрабатываемое приложение состоит из нескольких процедур, причем в некоторых из них необходимо создать по обработчику ошибок, бывает более удобно для сокращения программы и большей ясности структуры кода написать отдельную процедуру с обработчиком всех ошибок.

Перехват ошибок с предложением пользователю выбора действий

При перехвате ошибок можно выводить на экран не только сообщение об ошибке, но и рекомендации пользователю с предложением различных вариантов действий. Например, в следующем коде (листинг 11.7) пользователь

информируется о возникшей ошибке, и ему предлагается либо повторить ввод, либо автоматически установить значения, используемые по умолчанию, либо прекратить выполнение задачи.

Листинг 11.7. Перехват ошибок с предложением пользователю выбора действий

```
Private Sub cmdOK_Click()
   Dim x As Double, y As Double
   On Error GoTo ErrorHandler
   x = CDbl(txtNumber.Text)
   y = 1 / x
ReturnPoint:
   txtInverse.Text = CStr(y)
Exit Sub

ErrorHandler:
   Dim m As Integer
   Select Case Err.Number
      Case 6
          m = MsgBox("Переполнение." & vbCr & _
                     "Установить значение по умолчанию" & vbCr & _
                     "или повторить ввод?", vbQuestion + vbYesNo)
          If m = vbYes Then
            txtNumber.Text = 1
            y = 1
            txtNumber.SetFocus
            Resume ReturnPoint
         End If
         If m = vbNo Then
            txtNumber.SetFocus
            Exit Sub
         End If
      Case 11
         m = MsgBox("На ноль делить нельзя. Повторить ввод?", _
                    vbQuestion + vbYesNo)
         If m = vbYes Then
            txtNumber.SetFocus
            Exit Sub
         End If
         If m = vbNo Then
            Unload Me
         End If
```

```
      Case 13
         m = MsgBox("Должно быть число. Повторить ввод?", _
                    vbQuestion + vbYesNo)
         If m = vbYes Then
            txtNumber.SetFocus
            Exit Sub
         End If
         If m = vbNo Then
            Unload Me
         End If
      Case Else
         MsgBox "Непредвиденная ошибка"
         Exit Sub
   End Select
End Sub
```

Построчный перехват ошибок

Многие методы, используемые в VBA, возвращают дополнительное значение, которое информирует об успешной или неудачной их реализации. Например, метод `GoalSeek` объекта `Range` возвращает значение `False`, если средство Подбор параметра не смогло найти подходящего решения:

```
If Not Range("B1").GoalSeek(0, Range("A1")) Then
   MsgBox "Проблемы"
End If
```

Нечто подобное можно создавать и в VBA, обрабатывая ошибки в каждой строке, где они могут возникнуть. При этом удобно воспользоваться той особенностью переменной типа `Variant`, что она может хранить значения типа `Error`, присвоенные ей с помощью функции преобразования типов `CVErr`. Как раз эта идеология и реализуется в предлагаемом варианте построчного перехвата ошибок приложения Обратное значение, которое рассматривается с разных сторон в данной главе (листинг 11.8).

Листинг 11.8. Обработчик ошибок

```
Private Sub cmdOK_Click()
   Dim X, Y
   X = txtNumber.Text
   If Trans(X) = False Then
      MsgBox "Проблемы"
      Exit Sub
   End If
```

```
   Y = InverseValue(X)
   If Trans(Y) = False Then
      MsgBox "Проблемы продолжаются"
      Exit Sub
   End If
   txtInverse.Text = Y
End Sub

Function InverseValue(X)
   Dim Y
   On Error Resume Next
   Y = 1 / X
   If Err.Number = 0 Then
      InverseValue = Y
   Else
      InverseValue = CVErr(Err.Number)
   End If
End Function

Function Trans(X)
   If IsNumeric(X) Then
      Trans = X
   Else
      Trans = False
   End If
End Function
```

Отладка программ

При написании программ пользователь, независимо от его опыта, допускает те или иные ошибки. Кто-то из мудрых совершенно верно подметил, что только тот, кто ничего не делает, не совершает ошибок. Позвольте перефразировать эту жизненную мудрость для изучаемого в данной книге предмета следующим образом: в программах не делал ошибок только тот, кто никогда не писал кода. Итак, ошибки — это объективная неизбежность или реальное воплощение этой неизбежности. Какие же бывают ошибки и как с ними бороться? Условно ошибки можно поделить на три типа: ошибки компиляции, выполнения и логические ошибки.

Ошибки компиляции

Ошибки компиляции возникают, если Visual Basic не может интерпретировать введенный код. Например, при некорректном вводе числа скобок, неправильном имени, неполном вводе инструкции и т. д. Некоторые из этих

ошибок обнаруживаются Visual Basic при завершении набора строки с инструкцией в редакторе кода нажатием клавиши <Enter>. Строка, в которой содержится ошибка, выделяется красным цветом, и на экране отображается диалоговое окно с сообщением о возможной причине, вызвавшей ошибку (рис. 11.3).

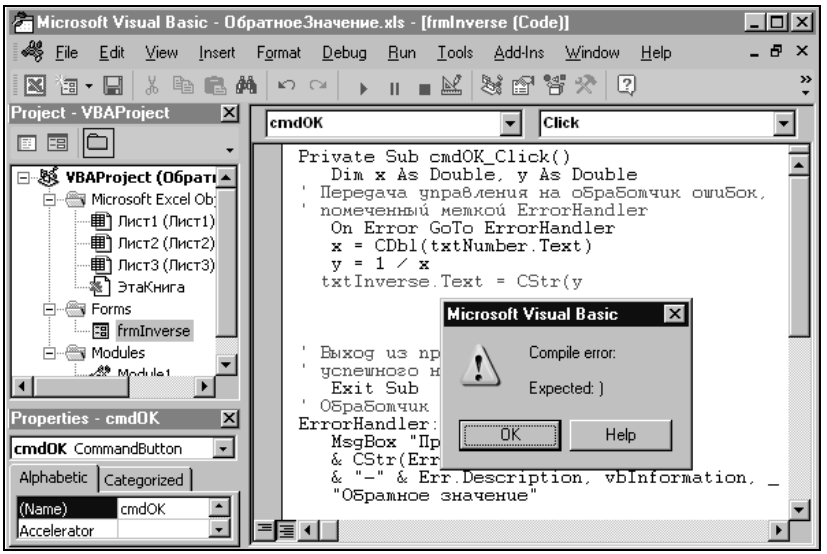


Рис. 11.3. Ошибка компиляции, обнаруженная при вводе инструкции

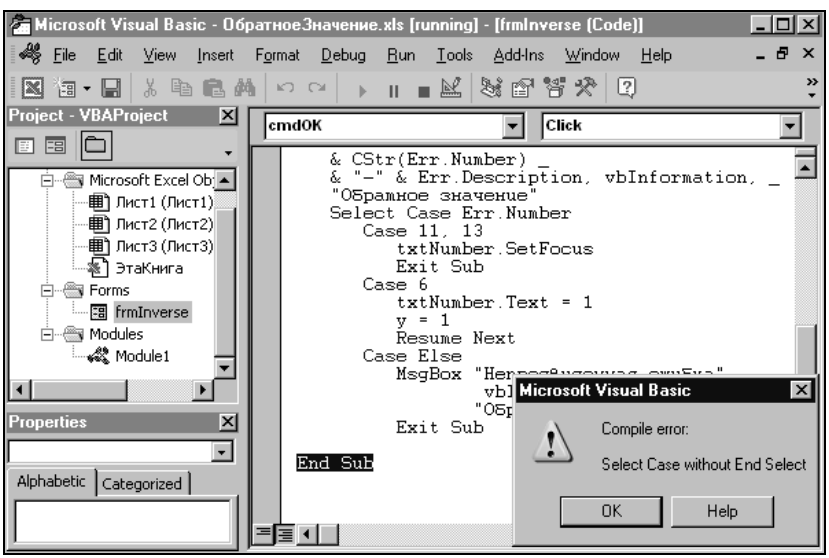


Рис. 11.4. Сообщение об ошибке компиляции в диалоговом окне **Microsoft Visual Basic**

Другие ошибки компиляции обнаруживаются перед выполнением программы. Отметим, что VBA каждый раз автоматически компилирует программу при ее запуске на выполнение. В этом случае предполагаемое местоположение ошибки выделяется синим цветом, и на экране отображается диалоговое окно **Microsoft Visual Basic** с сообщением о возможной причине, вызвавшей ошибку, например опущена инструкция `End Select` в операторе `Select` (рис. 11.4).

Ошибки выполнения

Ошибки выполнения возникают после успешной компиляции программы при ее выполнении. Причинами таких ошибок могут быть, например:

- некорректная информация при считывании файла с диска;
- некорректные данные, введенные пользователем, например требуется число, а пользователь вводит строковую информацию;
- некорректность вычислений, например деление на ноль;

и т. д.

В этом случае на экране отображается диалоговое окно **Microsoft Visual Basic** с сообщением о номере ошибки и возможной причине, ее вызвавшей (рис. 11.5).

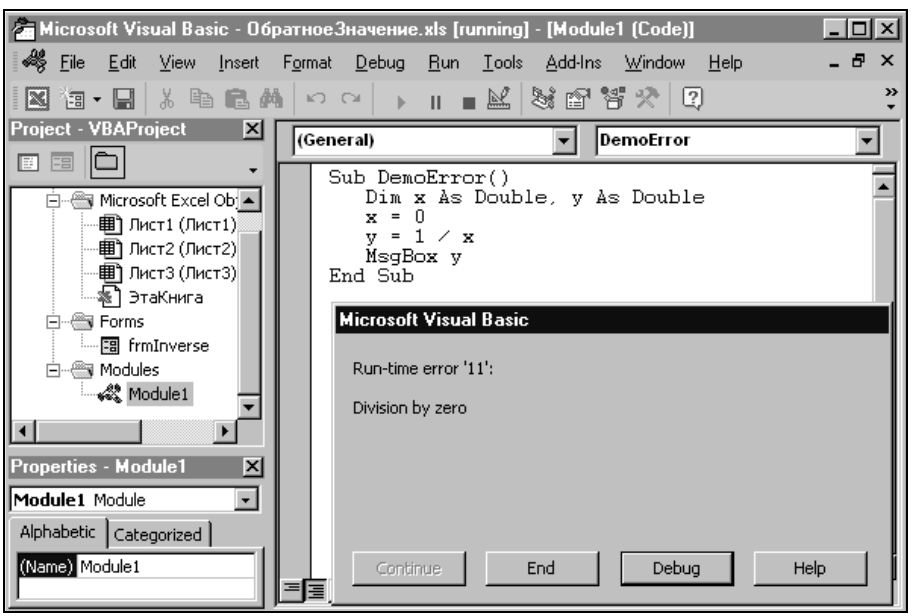


Рис. 11.5. Сообщение об ошибке выполнения в диалоговом окне **Microsoft Visual Basic**

Если в диалоговом окне **Microsoft Visual Basic** нажать кнопку **Debug**, то в строке модуля желтым цветом будет выделена строка, вызвавшая ошибку, на которой выполнение программы было прервано. Кроме того, эта строка будет помечена стрелочкой. VBA перейдет в режим прерывания (рис. 11.6).

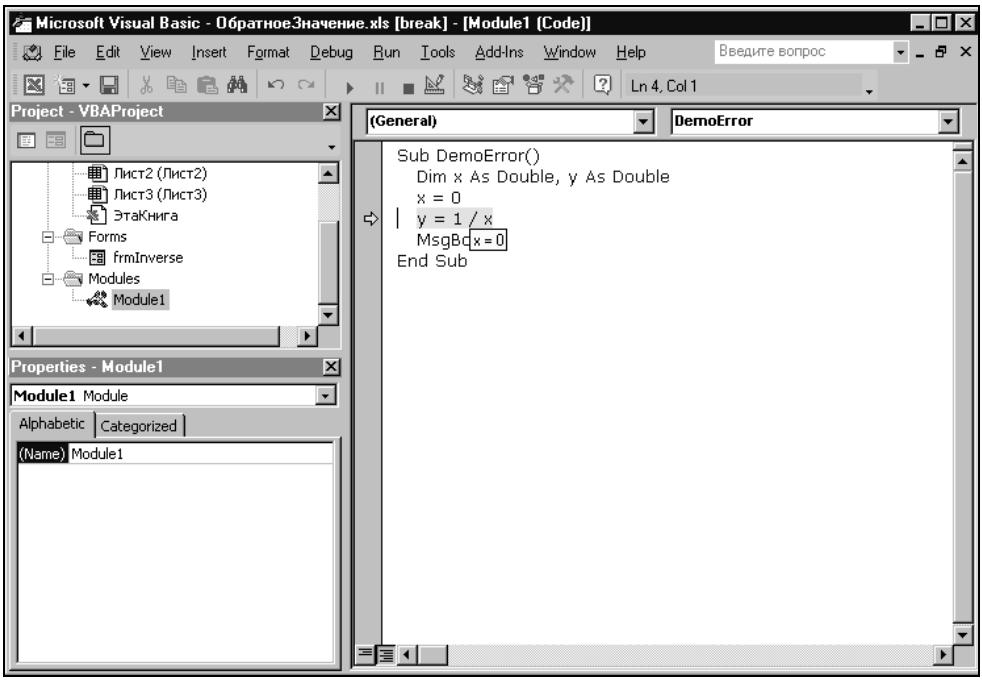


Рис. 11.6. Редактор кода в режиме прерывания

Одним из удобств режима прерывания является возможность узнать текущее значение переменных и свойств, для чего достаточно расположить указатель мыши на имени свойства или переменной. Это вызовет появление всплывающей подсказки с текущим значением переменной или свойства. В данном случае (см. рис. 11.6) видно, что значение переменной *y* равно 0, что и вызвало ошибку. Для задания режима вывода всплывающей подсказки с текущими значениями данных должен быть установлен флажок **Auto Data Tips** вкладки **Editor** диалогового окна **Options** вызываемого командой **Tools | Options**.

Кроме режима прерывания, приложение может находиться в режиме разработки и режиме выполнения. В режиме разработки, собственно, и создается приложение — конструируются формы, набирается код. В режиме выполнения приложение получает управление, и мы взаимодействуем с ним так же, как это будет делать и пользователь нашего приложения. Переключение

между режимами работы удобно производить при помощи трех кнопок панели инструментов **Standard** (кстати, они также включены в панель инструментов **Debug**), перечисленных в табл. 11.5.

Таблица 11.5. *Кнопки панели инструментов* ***Standard****, управляющие режимом работы программы*

Кнопка	Название	Описание
	**Start**	Доступна в режиме конструирования. Переключает в режим выполнения. В режиме прерывания она также доступна, но играет роль кнопки **Continue**
	**Break**	Доступна в режиме выполнения. Переключает в режим прерывания
	**End**	Доступна в режиме выполнения. Переключает в режим разработки

Логические ошибки

Логические ошибки труднее всего обнаружить и устранить. Эти ошибки не приводят к прерыванию выполнения программы, т. е. визуально все идет гладко и выглядит так, как будто программа работает безупречно. Но это только кажущаяся идиллия, поскольку программа выдает неверные результаты. Локализация логических ошибок связана с тщательным анализом алгоритма программы с привлечением средств отладки VBA.

Инструкция *Option Explicit*

Простейшим средством предотвращения случайных ошибок является использование инструкции `Option Explicit`. Эта инструкция предписывает объявлять все переменные, встречающиеся в программе. Использование инструкции `Option Explicit` позволяет избежать следующей трудно отслеживаемой ошибки. Предположим, что в программе используется переменная с именем `Ссуда`, а при наборе имени этой переменной где-то в программе вместо русской буквы "`с`" по ошибке набрана латинская буква "`c`". Визуально эти имена ничем не отличаются друг от друга, но воспринимаются программой как имена различных переменных. Если бы была использована инструкция `Option Explicit`, и переменная `Ссуда` была бы объявлена, то компилятор указал бы на переменную `Ссуда` с латинской буквой "`c`" как на необъявленную, и эта трудно отслеживаемая ошибка была бы быстро найдена.

Пошаговое выполнение программ

Редактор Visual Basic позволяет осуществлять пошаговое выполнение программы. Такой режим можно задать либо при помощи панели инструментов **Debug**, либо из меню **Debug**, которое включает команды и соответствующие комбинации клавиш (рис. 11.7). Если панель инструментов **Debug** не отображена на экране, то ее можно отобразить командой **View | ToolBars | Debug**.

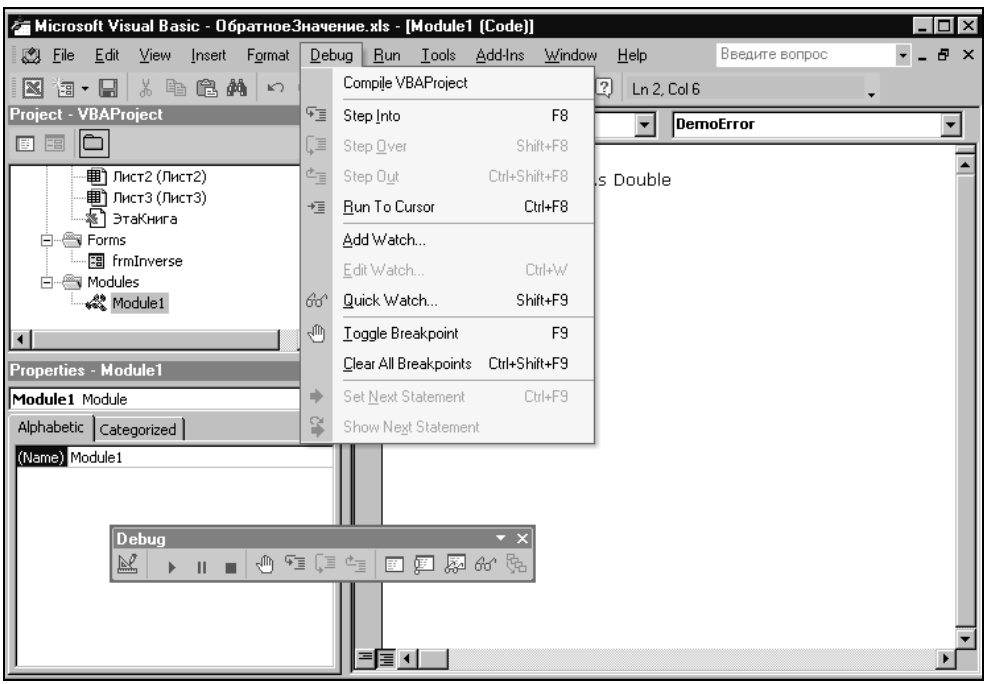


Рис. 11.7. Панель инструментов **Debug** и раскрывающееся меню **Debug**

Для выполнения программы в пошаговом режиме используются четыре команды:

- команда **Debug | Step Into** либо кнопка панели инструментов **Debug** осуществляют последовательную, шаг за шагом, отладку всей программы, включая процедуры, вызываемые в программе;
- команда **Debug | Step Over** либо кнопка панели инструментов **Debug** осуществляют последовательную, шаг за шагом, отладку всей программы, но не заходя в процедуры, вызываемые в программе. Если встречается процедура, то она выполняется целиком, а не шаг за шагом, как это делается в команде **Debug | Step Into**;

- команда **Debug | Step Out** либо кнопка панели инструментов **Debug** завершают выполнение текущей процедуры и останавливаются на следующей инструкции программы, откуда процедура была вызвана;
- команда **Debug | Run to Cursor** выполняет программу до инструкции, помеченной курсором.

Точка прерывания

VBA приостанавливает выполнение программы перед строкой кода, содержащей точку прерывания, и переключается в режим прерывания. Точка прерывания устанавливается или снимается командой **Debug | Toggle Breakpoint** либо кнопкой панели инструментов **Debug**. В модуле точки прерывания выделяются полосой кирпичного цвета и кругом того же цвета (рис. 11.8).

В одном проекте может быть несколько точек прерывания. Все инструкции, расположенные выше, между и ниже точек прерывания, выполняются в обычном режиме.

Одновременно снять все точки прерывания можно командой **Debug | Clear All Breakpoint**.

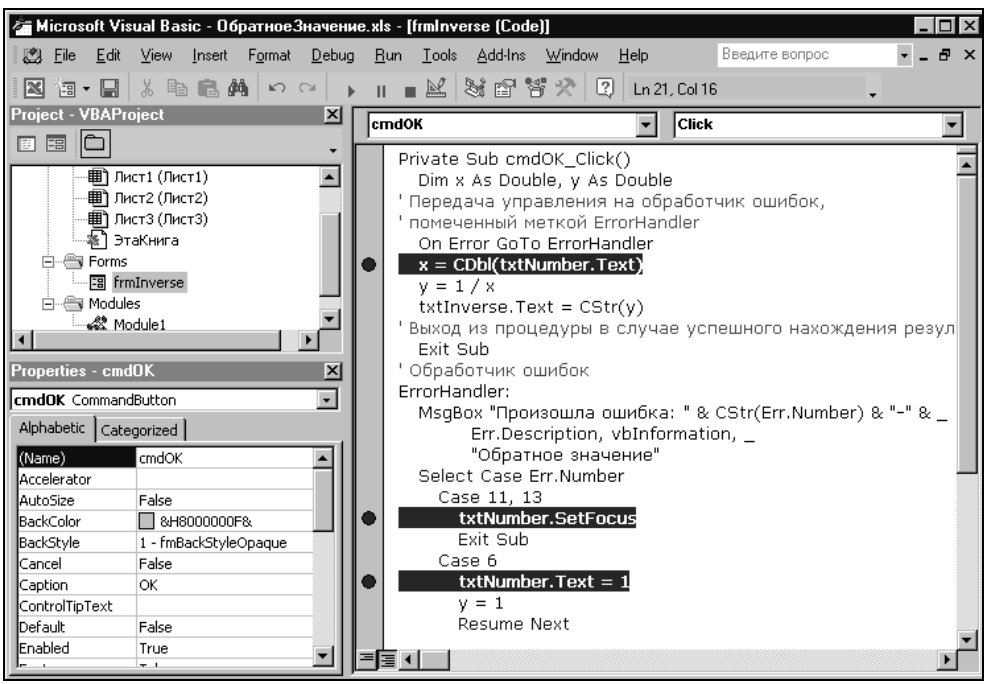


Рис. 11.8. Точки прерывания

Вывод значений свойств и переменных

Одним из удобств режима отладки является возможность узнать текущее значение переменных и свойств, для чего, как и в режиме прерывания, достаточно расположить указатель мыши на имени свойства или переменной. Это вызовет появление всплывающей подсказки с текущим значением переменной или свойства (см. рис. 11.6).

Окно *Watches*

Другим способом отслеживания текущих значений данных является использование диалогового окна **Watches**, которое отображается на экране либо командой **View | Watch Window**, либо командой **Debug | Quick Watch** (рис. 11.9). Диалоговое окно **Watches** позволяет одновременно отображать текущие значения нескольких переменных или свойств. Команда **Debug | Add Watch** добавляет новые контрольные значения в диалоговое окно **Watches**.

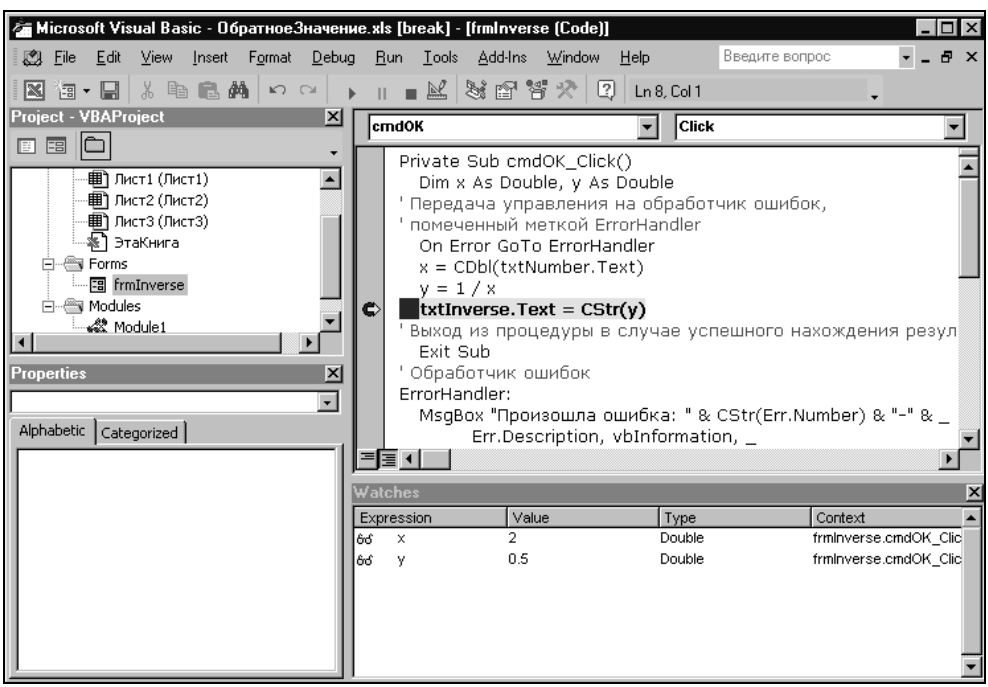


Рис. 11.9. Окно **Watches**

Удаление контрольного значения из диалогового окна производится его выделением и нажатием клавиши <Delete>.

Окно *Locals*

Окно **Locals**, отображаемое на экране командой **View | Locals Window**, выводит значения всех переменных текущей процедуры, а не только специально выбранных, как это происходит в окне **Watches**. Внешний же вид и структура обеих окон **Watches** и **Locals** одна и та же.

Окно *Immediate*

Окно **Immediate**, отображаемое на экране командой **View | Immediate Window**, предоставляет пользователю следующие возможности:

- набирать и вычислять отдельные инструкции VBA. Для этого достаточно в окне **Immediate** ввести соответствующую инструкцию и нажать клавишу <Enter>. Единственным ограничением на инструкцию является то, что она должна быть набрана в одну строку. Например:

```
s = 0: For i=1 to 5: s = s + i ^ 2: Next i: MsgBox s
```

- определять текущие значения переменных и свойств. Для этого в окне **Immediate** надо набрать вопросительный знак, имя переменной или свойства и нажать клавишу <Enter>. Например:

```
?x
```

- устанавливать новые текущие значения переменных. Для этого в окне **Immediate** надо набрать имя переменной, знак "равно" и новое значение переменной:

```
x = 15
```

- определять значения встроенных в VBA констант. Для этого в окне **Immediate** надо набрать вопросительный знак, имя константы и нажать клавишу <Enter>. Например:

```
? vbKeyReturn
```

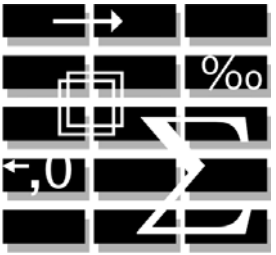
Программный способ вывода значений в окно *Immediate*

Существует также программный способ вывода значения свойств и переменных в диалоговое окно **Immediate** при помощи метода `Print` объекта `Debug`. Ниже приведен пример программного способа вывода значений переменных в окно **Immediate** (листинг 11.9).

Листинг 11.9. Программный способ вывода значений в окно Immediate

```
Dim n As Integer, Res As Integer
Randomize
For n = 1 To 10
   Res = Int(6 * Rnd()) + 1
   Debug.Print n, Res
Next
```

ГЛАВА 12



Помощник MS Office

Объект *Assistant*

В MS Office имеется очень удобное средство — помощник, которое максимально приспособлено для отображения справочной информации и подсказок пользователю. Вся информация о помощнике инкапсулируется в объекте `Assistant`, который возвращается свойством `Assistant` объекта `Application`.

Свойства объекта *Assistant*

Свойства объекта `Assistant` позволяют контролировать поведения помощника от спецификации проигрываемой анимации до разделов отображаемой справки. Эти свойства перечислены в табл. 12.1

Таблица 12.1. *Свойства объекта* `Assistant`

Свойство	Описание
`Animation`	Устанавливает используемую помощником мультипликацию
`AssistWithAlerts`	Определяет, надо ли отображать сообщения помощника. Это свойство эквивалентно установке флажка **отображать сообщения помощника** вкладки **Параметры** окна **Помощник** (рис. 12.1)
`AssistWithHelp`	Определяет, надо ли отображать помощника при нажатии клавиши <F1>. Это свойство эквивалентно установке флажка **вызывать при нажатии клавиши F1** вкладки **Параметры** окна **Помощник** (рис. 12.1)

Таблица 12.1 *(продолжение)*

Свойство	Описание
`BalloonError`	Возвращает число, идентифицирующее последнюю ошибку, произошедшую при работе с помощником. Допустимыми значениями являются следующие константы: `msoBalloonErrorBadCharacte`, `msoBalloonErrorBadPictureRer`, `msoBalloonErrorBadReference`, `msoBalloonErrorButtonlessModal`, `msoBalloonErrorButtonModeless`, `msoBalloonErrorCharNotTopmostForModa`, `msoBalloonErrorCOMFailure`, `msoBalloonErrorNone`, `msoBalloonErrorOther`, `msoBalloonErrorOutOfMemory`, `msoBalloonErrorTooBig`, `msoBalloonErrorTooManyControls`
`FeatureTips`	Это свойство эквивалентно установке флажка **об использовании возможностей программы** группы **Отображать советы** вкладки **Параметры** окна **Помощник** (рис. 12.1)
`FileName`	Имя искомого файла
`GuessHelp`	Это свойство эквивалентно установке флажка **подбирать разделы справки** вкладки **Параметры** окна **Помощник** (рис. 12.1)
`HighPriorityTips`	Это свойство эквивалентно установке флажка **отображать только важные советы** вкладки **Параметры** окна **Помощник** (рис. 12.1)
`KeyboardShortcutTips`	Это свойство эквивалентно установке флажка **о сочетаниях клавиш** группы **Отображать советы** вкладки **Параметры** окна **Помощник** (рис. 12.1)
`Left`, `Top`	Координаты левого верхнего угла окна помощника
`MouseTips`	Это свойство эквивалентно установке флажка **об операциях с мышью** группы **Отображать советы** вкладки **Параметры** окна **Помощник** (рис. 12.1)
`MoveWhenInTheWay`	Это свойство эквивалентно установке флажка **не закрывать данные** группы **Использовать помощника** вкладки **Параметры** окна **Помощник** (рис. 12.1)
`Name`	Имя помощника
`NewBalloon`	Возвращает объект `Balloon`, который представляет окно, в котором помощник отображает информацию
`On`	Определяет, активизирован ли помощник

Таблица 12.1 *(окончание)*

Свойство	Описание
`Reduced`	Устанавливает, отображается ли помощник уменьшенного размера
`SearchWhenProgramming`	Определяет, отображать ли помощника в то время, когда пользователь набирает код
`Sounds`	Это свойство эквивалентно установке флажка **звуковое сопровождение** группы **Использовать помощника** вкладки **Параметры** окна **Помощник** (рис. 12.1)
`TipOfDay`	Это свойство эквивалентно установке флажка **отображать полезный совет при запуске** группы **Отображать советы** вкладки **Параметры** окна **Помощник** (рис. 12.1)
`Visible`	Устанавливает видимость помощника

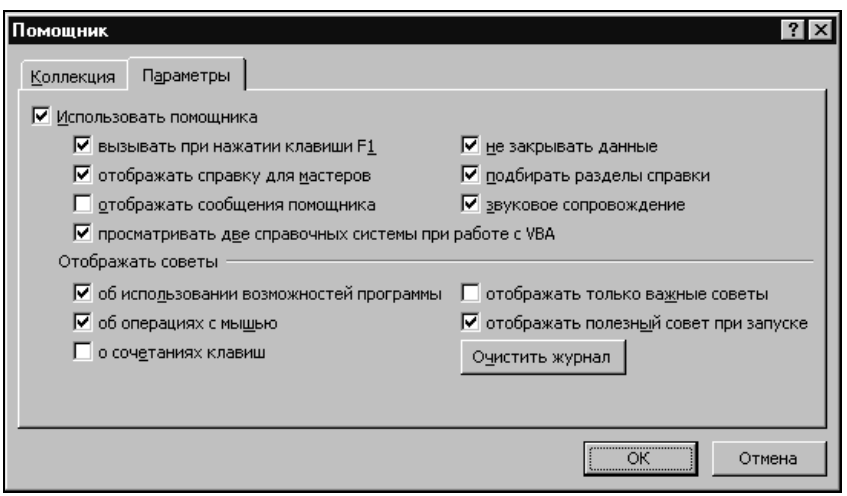


Рис. 12.1. Вкладка **Параметры** окна **Помощник**

Методы объекта *Assistant*

Методы объекта `Assistant` позволяют управлять жизненным циклом помощника от его инициализации до закрытия. Эти методы перечислены в табл. 12.2.

Таблица 12.2. *Методы объекта* `Assistant`

Метод	Описание
`ActivateWizard`	Активизирует мастера помощника. Используется совместно с методом `StartWizard`
`DoAlert`	Определяет нажатую пользователем кнопку
`EndWizard`	Высвобождает из памяти ресурсы, отведенные под переменную, созданную методом `StartWizard` для идентификации сеанса
`Help`	Отображает окно помощника для получения стандартной справки по MS Office
`Move`	Перемещает помощника в специфицированное местоположение
`ResetTips`	Переустанавливает подсказки, отображаемые в окне помощника
`StartWizard`	Запускает помощника и создает переменную типа `Long`, идентифицирующую сеанс

Объект *Balloon*

Структура помощника, его внешний вид конструируется при помощи объекта `Balloon`. Новый объект `Balloon` создается свойством `NewBalloon` объекта `Assistant`. В объекте `Balloon`, как в родительском объекте, могут размещаться надписи (объекты `BalloonLabel`) и флажки (объекты `BalloonCheckBox`), которые образуют семейства `BalloonLabels` и `BalloonCheckBoxes`.

Основными свойствами объекта `BalloonCheckBox` являются `Checked` и `Text`, первое из которых определяет, установлен ли флажок, а второе задает текст, отображаемый рядом с флажком. Основным свойствам объекта `BalloonLabel` является свойство `Text`.

У объекта `Balloon` имеется большая коллекция свойств, задающих интерфейс помощника, и ряд методов, определяющих его функционирование. В табл. 12.3 и 12.4 как раз и перечислены свойства и методы объекта `Balloon`.

Таблица 12.3. *Свойства объекта* `Balloon`

Свойство	Описание
`Animation`	Задает анимацию. Допустимыми значениями являются следующие константы: `msoAnimationCheckingSomething`, `msoAnimationGetTechy`, `msoAnimationListensToComputer`, `msoAnimationSearching`, `msoAnimationThinking`, `msoAnimationWorkingAtSomething` и `msoAnimationWritin`

Таблица 12.3 *(окончание)*

Свойство	Описание
BalloonType	Задает окна помощника. Допустимыми значениями являются следующие константы: msoBalloonTypeBullets (маркированный список), msoBalloonTypeButtons (надписи выполняют роль кнопок) и msoBalloonTypeNumbers (нумерованный список)
Button	Устанавливает тип кнопок, отображаемых в окне. Допустимыми значениями являются следующие константы: msoButtonSetAbortRetryIgnore, msoButtonSetBackClose, msoButtonSetBackNextClose, msoButtonSetBackNextSnooze, msoButtonSetCancel, msoButtonSetNextClose, msoButtonSetNone, msoButtonSetOK, msoButtonSetOkCancel, msoButtonSetRetryCancel, msoButtonSetSearchClose, msoButtonSetTipsOptionsClose, msoButtonSetYesAllNoCancel, msoButtonSetYesNo и msoButtonSetYesNoCancel
Callback	Задает имя процедуры, запускаемой из помощника
Checkboxes	Возвращает семейство BalloonCheckBoxes
Heading	Заголовок окна
Icon	Значок, отображаемый в верхнем левом углу помощника. Допустимыми значениями являются следующие константы: msoIconAlert, msoIconAlertCritical, msoIconAlertInfo, msoIconAlertQuery, msoIconAlertWarning, msoIconNone и msoIconTip
Labels	Возвращает семейство BalloonLabels
Mode	Задает, является окно моно- или полимодальным. Допустимыми значениями являются следующие константы: msoModeAutoDown, msoModeModal и msoModeModeless
Name	Имя объекта
Private	Возвращает целое число, идентифицирующее объект, запустивший на выполнение процедуру при помощи свойства Callback
Text	Задает текст, отображаемый в окне помощника

Таблица 12.4. *Методы объекта* Balloon

Метод	Описание
Close	Закрывает объект Balloon
SetAvoidRectangle	Запрещает отображение окна помощника в специфицированном месте на экране
Show	Отображает окно помощника

Помощник с надписями и вызовом процедуры

Помощник создается свойством `NewBalloon` объекта `Assistant`. Свойство `Heading` задает заголовок помощника, а свойство `Labels` возвращает семейство надписей помощника. Для того чтобы при выборе элемента помощника вызывалось выполнение какого-либо действия, надо свойством `Callback` ассоциировать с помощником процедуру.

Процедура, вызываемая свойством `Callback`, зависит от того, как был вызван помощник.

Если помощник был вызван в полимодальном режиме, то эта процедура должна иметь три параметра: первый из них возвращает ссылку на объект `Balloon`, вызвавший процедуру. Второй параметр возвращает идентификатор нажатой кнопки помощника, а третий параметр — целое число, определенное свойством `Private` и идентифицирующее помощника.

Если помощник был вызван мастером помощника, то процедура должна иметь два параметра: первый из них возвращает идентификатор нажатой кнопки помощника, а второй — целое число, идентифицирующее помощника.

Отображается помощник методом `Show`, а закрывается — методом `Close`.

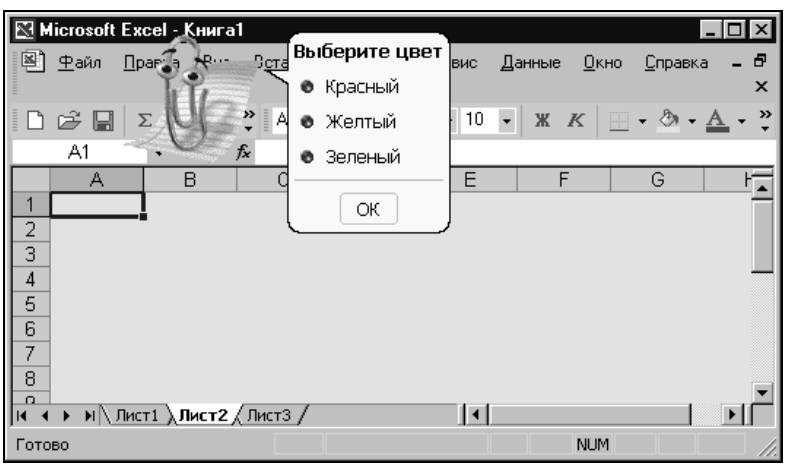


Рис. 12.2. Помощник с надписями

Следующий код (листинг 12.1) демонстрирует работу с помощником. При щелчке правой кнопкой на ячейке **A1** любого рабочего листа отображается окно помощника с предложением изменить цвет фона рабочего листа (рис. 12.2). Выбрав требуемый цвет и нажав кнопку **OK**, пользователь перекрашивает рабочий лист. Комбинация кнопок, отображаемая в окне по-

мощника, устанавливается свойством Button. Значение этого свойства в данном случае установлено равным msoButtonSetOK, поэтому и имеется только одна кнопка **OK**.

Листинг 12.1, *а*. Помощник с надписями. Код из модуля ЭтаКнига

```
Private Sub Workbook_SheetBeforeRightClick(ByVal Sh As Object, _
      ByVal Target As Range, Cancel As Boolean)
   If Target.Address = "$A$1" Then
      Cancel = True
      SelectColor
   End If
End Sub

Sub SelectColor()
   Dim bln As Balloon
   Set bln = Assistant.NewBalloon
   With bln
      .Heading = "Выберите цвет"
      .Labels(1).Text = "Красный"
      .Labels(2).Text = "Желтый"
      .Labels(3).Text = "Зеленый"
      .Button = msoButtonSetOK
      .BalloonType = msoBalloonTypeButtons
      .Mode = msoModeModeless
      .Callback = "SetColor"
      .Show
   End With
End Sub
```

Листинг 12.1, *б*. Помощник с надписями. Стандартный модуль

```
Private Sub SetColor(bln As Balloon, lbtn As Long, lPriv As Long)
   Assistant.Animation = msoAnimationGestureDown
   Select Case lbtn
      Case 1
         ActiveSheet.Cells.Interior.Color = RGB(255, 0, 0)
      Case 2
         ActiveSheet.Cells.Interior.Color = RGB(255, 255, 0)
      Case 3
         ActiveSheet.Cells.Interior.Color = RGB(0, 255, 0)
    End Select
    bln.Close
End Sub
```

Помощник с флажками

Флажки на помощнике реализуются объектами `CheckBox`. Состояние флажка возвращается свойством `Checked`, а текст, отображаемый рядом с флажком, — свойством `Text`. В следующим примере (листинг 12.2) создается помощник, который отображается при нажатии правой кнопки мыши. Он позволяет считывать и устанавливать стиль шрифта выделенного диапазона. В окне помощника имеются две кнопки: **OK** и **Отмена**, что обеспечивается установкой свойству `Button` значения `msoButtonSetOkCancel` (рис. 12.3). Какая из этих кнопок нажата, идентифицируется параметром `lbtn` процедуры `SetStyle`, назначенной свойством `Callback` объекта `Balloon`. Если этот параметр возвращает значение −1, то нажата кнопка **OK**, если −2, то кнопка **Отмена**.

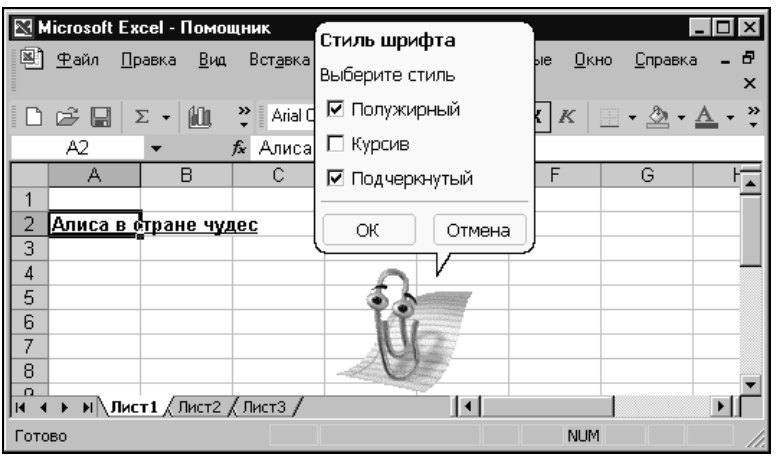


Рис. 12.3. Помощник с флажками

Листинг 12.2, *а*. Помощник с флажками. Модуль рабочего листа

```
Private Sub Worksheet_BeforeRightClick(ByVal Target As Range, _
      Cancel As Boolean)
   Cancel = True
   ShowAssistant
End Sub

Private Sub ShowAssistant()
   Dim bln As Balloon
   Set bln = Assistant.NewBalloon
   With bln
      .Heading = "Стиль шрифта"
```

```
      .Text = "Выберите стиль"
      .CheckBoxes(1).Text = "Полужирный"
      .CheckBoxes(2).Text = "Курсив"
      .CheckBoxes(3).Text = "Подчеркнутый"
      .Button = msoButtonSetOkCancel
      .CheckBoxes(1).Checked = Selection.Font.Bold
      .CheckBoxes(2).Checked = Selection.Font.Italic
      .CheckBoxes(3).Checked = Selection.Font.Underline
      .Mode = msoModeModeless
      .Callback = "SetStyle"
      .Show
   End With
End Sub
```

Листинг 12.2, *б*. Помощник с флажками. Стандартный модуль

```
Sub SetStyle(bln As Balloon, lbtn As Long, lPriv As Long)
   Select Case lbtn
      Case -1 ' Кнопка OK
         Selection.Font.Bold = bln.CheckBoxes(1).Checked
         Selection.Font.Italic = bln.CheckBoxes(2).Checked
         Selection.Font.Underline = bln.CheckBoxes(3).Checked
         bln.Close
      Case -2 ' Кнопка Отмена
         bln.Close
   End Select
End Sub
```

Добавление значка в заголовок помощника

Свойство `Icon` объекта `Balloon` позволяет добавить один из стандартных значков в заголовок помощника. Допустимыми значениями являются следующие константы: `msoIconAlert`, `msoIconAlertCritical`, `msoIconAlertInfo`, `msoIconAlertQuery`, `msoIconAlertWarning`, `msoIconNone` и `msoIconTip`. В приведенном ниже примере (листинг 12.3) окно помощника отображается при активизации рабочего листа, а в заголовке выводится значок — вопросительный знак (рис. 12.4). В этом коде имеется еще одна особенность, на которую следует обратить внимание. Нажатая кнопка идентифицируется не процедурой, вызываемой свойством `Callback`, а по целому числу, возвращаемому методом `Show`, который и отображает окно помощника.

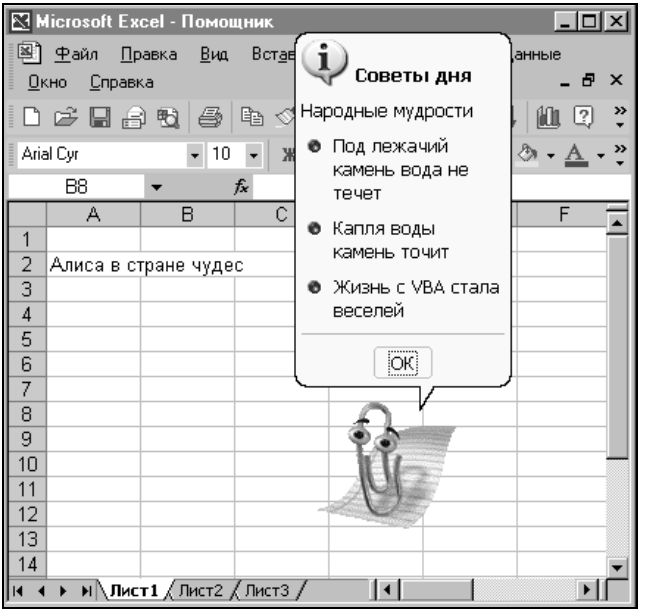


Рис. 12.4. Добавление значка в заголовок помощника

Листинг 12.3. Добавление значка в заголовок помощника. Модуль рабочего листа

```
Private Sub Worksheet_Activate()
   AssistantShow
End Sub

Sub AssistantShow()
   Dim idx As Integer
   Dim bln As Balloon
   Set bln = Assistant.NewBalloon
   Assistant.Animation = msoAnimationSaving
   With bln
      .BalloonType = msoBalloonTypeButtons
      .Icon = msoIconAlertInfo
      .Heading = "Советы дня"
      .Text = "Народные мудрости"
      .Labels(1).Text = "Под лежачий камень вода не течет"
      .Labels(2).Text = "Капля воды камень точит"
      .Labels(3).Text = "Жизнь с VBA стала веселей"
          idx = .Show
   End With
```

```
   AssistantResponse idx
End Sub

Sub AssistantResponse(idx As Integer)
   Select Case idx
      Case 1
         MsgBox "Вывод: пора начать читать книгу по MS Excel 2002"
      Case 2
         MsgBox "Вывод: если приложить достаточно усилий," & vbCr & _
               "то можно разобраться с любой проблемой"
      Case 3
         MsgBox "Что верно, то верно. MS Excel 2002 с VBA — это мощь"
      Case Else
         MsgBox "Ничего не выбрали"
   End Select
End Sub
```

Добавление значка в надписи и флажки помощника

Значки на основе растровых файлов можно добавлять в те элементы помощника, у которых имеется свойство `Text`: в текстовую строку самого помощника, в надписи и флажки (рис. 12.5). При этом для спецификации файла, из которого загружается растровое изображение, используется следующий синтаксис:

```
{type location sizing_factor}
```

- *type* — задает тип файла. Допустимы значения `bmp` и `wmf`;
- *location* — определяет местоположение файла;
- *sizing_factor* — задает ширину для WMF-файлов.

Как это делается, продемонстрировано на следующем примере (листинг 12.4). В нем при щелчке правой кнопкой мыши отображается окно с предложением одной из двух альтернатив, выбор которых приводит к вводу соответствующего текста в назначенную ячейку.

Листинг 12.4. Добавление значка в надписи и флажки помощника. Модуль рабочего листа

```
Private Sub Worksheet_BeforeRightClick(ByVal Target As Range, _
                                       Cancel As Boolean)
   Cancel = True
```

```
   ShowAssistant
End Sub

Private Sub ShowAssistant()
   Dim bmpCat As String, bmpDog As String
   Dim bln As Balloon
   Dim idx As Integer
   bmpCat = "{bmp c:\Cat.bmp}"
   bmpDog = "{bmp c:\Dog.bmp}"
   Set bln = Assistant.NewBalloon
   With bln
      .Heading = "Текст с картинками"
      .Labels(1).Text = "Кошка" & bmpCat
      .Labels(2).Text = "Собака" & bmpDog
      idx = .Show
   End With
   Select Case idx
      Case 1
         Selection.Value = "Кошка"
      Case 2
         Selection.Value = "Собака"
    End Select
End Sub
```

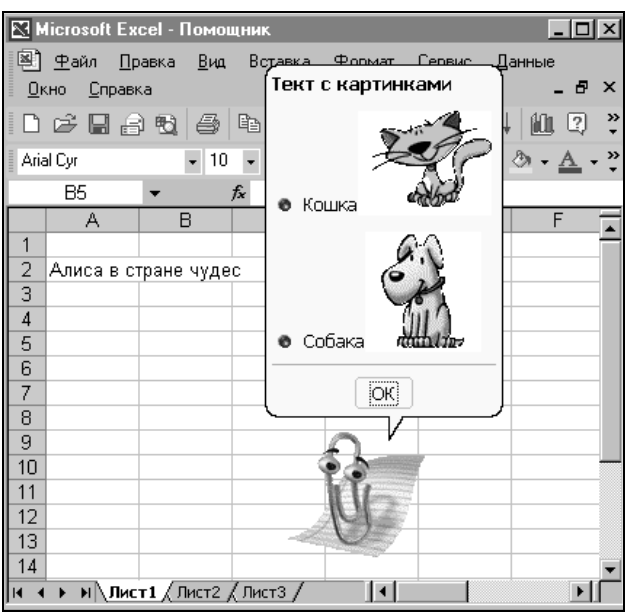


Рис. 12.5. Добавление значка в заголовок помощника

Раскрашивание помощника

У тех элементов помощника, которые имеют свойство `Text`: текстовая строка самого помощника, надписи и флажки, допустимо как раскрашивать фрагменты текста в 16 системных цветов, так и подчеркивать их. Для включения режима подчеркивания текста используется атрибут `{ul}` или `{ul 1}`, а для отключения режима подчеркивания — атрибут `{ul 0}`. Для окрашивания фрагмента текста применяется атрибут `{cf number}`, допустимые значения параметра *number*, задающего цвет, перечислены в табл. 12.5.

Таблица 12.5. *Допустимые значения параметра* `number`*, задающего цвет*

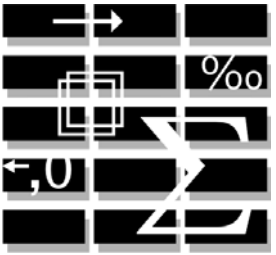
Постоянная цвета	Цвет	Постоянная цвета	Цвет
0	Черный	248	Серый
1	Темно-красный	249	Красный
2	Темно-зеленый	250	Зеленый
3	Темно-желтый	251	Желтый
4	Темно-голубой	252	Голубой
5	Темно-розовый	253	Розовый
6	Темно-синий	254	Синий
7	Темно-серый	255	Белый

Следующий код (листинг 12.6) демонстрирует, как можно создать раскрашенного помощника.

Листинг 12.6. Раскрашивание помощника

```
Sub DemoColors()
   With Assistant.NewBalloon
      .Heading = "Пример раскрашивания текста"
      .Text = "Это {cf 249}красный {cf 0}цвет." & vbCr _
       & "Это {cf 250}зеленый{cf 0}" & vbCr _
       & "{ul 1}А это подчеркнутый{ul 0}"
      .Show
   End With
End Sub
```

ГЛАВА 13



Объектная файловая система

VBA позволяет обрабатывать файлы, работать с дисками и каталогами двумя способами:

- посредством объектной файловой системы (File System Object Model, FSO);
- традиционным, использующим оператор `Open`.

FSO предоставляет разработчику возможность работы с файлами в ясной объектно-ориентированной манере. FSO опирается на библиотеку Microsoft Scripting Runtime Library (файл Scrrun.dll), которая совместно используется несколькими средами разработки и позволяет, в частности, расширить функции написания сценариев. Для применения этой библиотеки надо установить ссылку на Microsoft Scripting Runtime в диалоговом окне **References**, которое отображается на экране выбором команды **Project | References**. После этого можно приступить к программированию объектов, функционирование которых обеспечивается этой библиотекой. Основными объектами FSO являются:

`Dictionary`	`FileSystemObject`
`Drive`	`Folder`
`File`	`TextStream`

В заключение главы будет рассмотрен объект `FileSearch` и ассоциированные с ним объекты, предоставляющие удобные средства поиска файлов на диске.

Объект *FileSystemObject* и доступ к объектной файловой системе

Объект `FileSystemObject` является верхним объектом в иерархии FSO. Он открывает доступ к объектной файловой системе.

Создается объект `FileSystemObject` одним из двух способов:

- объявлением переменной типа `FileSystemObject`:

```
Dim fso As New FileSystemObject
```

- использованием функции `CreateObject` для создания объекта `FileSystemObject`:

```
Set fso = CreateObject("Scripting.FileSystemObject")
```

Свойства объекта `FileSystemObject` (табл. 13.1) позволяют получить доступ к семейству драйверов и каталогов, а методы объекта `FileSystemObject` (табл. 13.2) — производить широкий спектр операций над файлом от его создания до удаления.

Таблица 13.1. *Свойства объекта* `FileSystemObject`

Свойство	Описание
`Drives`	Возвращает семейство `Drives` всех дисков
`Files`	Возвращает семейство `Files` всех файлов из указанного каталога

Таблица 13.2. *Методы объекта* `FileSystemObject`

Метод	Описание
`BuildPath`	Создает полное имя по пути и имени файла
`CopyFile`	Копирует файл или файлы
`CreateFolder`	Создает каталог
`CreateTextFile`	Создает текстовый файл
`DeleteFile`	Удаляет файл
`DeleteFolder`	Удаляет каталог
`DriveExists`	Проверяет существование диска
`FileExists`	Проверяет существование файла
`FolderExists`	Проверяет существование каталога
`GetAbsolutePathName`	Возвращает полный путь
`GetBaseName`	Возвращает базовое имя компонента
`GetDrive`	Возвращает объект `Drive` по пути
`GetDriveName`	Возвращает имя диска по пути
`GetExtensionName`	Возвращает имя расширения

Таблица 13.2 *(окончание)*

Метод	Описание
`GetFile`	Возвращает объект `File` по пути
`GetFileName`	Возвращает имя файла по пути
`GetFolder`	Возвращает объект `Folder` по пути
`GetParentFolderName`	Возвращает имя родительского каталога
`GetSpecialFolder`	Возвращает один из системных каталогов
`GetTempName`	Возвращает случайно сгенерированное системой имя временного файла или каталога
`MoveFile`	Перемещает файл или группу файлов
`MoveFolder`	Перемещает каталог или группу каталогов
`OpenTextFile`	Открывает текстовый файл

Семейства *Drives, Folders* и *Files*

Семейства `Drives`, `Folders` и `Files` объектов `Drive`, `Folder` и `File` представляют собой семейства всех дисков, каталогов и файлов. В каждом из этих семейств имеется по два свойства `Item` и `Count`, возвращающих элемент семейства и их число. В семействе `Folders` также существует единственный метод `Add`, который добавляет каталог в это семейство.

Объект *Drive*

Объект `Drive` инкапсулирует в себе информацию о диске, включая наличие свободного пространства и имя диска, которую можно получить при помощи его свойств, перечисленных в табл. 13.3.

Таблица 13.3. *Свойства объекта* `Drive`

Свойство	Описание
`AvailableSpace`	Возвращает доступное пространство на диске
`DriveLetter`	Возвращает метку диска
`DriveType`	Возвращает тип диска
`FileSystem`	Возвращает тип файловой системы
`FreeSpace`	Возвращает объем свободного пространства на диске

Таблица 13.3 (окончание)

Свойство	Описание
IsReady	Определяет, готов ли диск для использования
Path	Возвращает путь к указанному файлу
RootFolder	Возвращает корневой каталог
SerialNumber	Возвращает серийный номер диска
TotalSize	Возвращает размер диска
VolumeName	Возвращает имя диска

Как проверить, существует ли диск

Для того чтобы проверить, существует ли диск, надо воспользоваться методом DriveExists объекта FileSystemObject. Этот метод возвращает значение True, если специфицированный диск существует, и значение False, если не существует (листинг 13.1).

Листинг 13.1. Проверка существования диска

```
Sub CheckDrive()
   Dim fs As New FileSystemObject
   Dim dn  As String
   dn = "c:"
   If fs.DriveExists(dn) Then
     MsgBox dn & " существует"
   Else
     MsgBox dn & " не существует"
   End If
End Sub
```

Как проверить, доступен ли диск

Для того чтобы проверить, доступен ли диск, например вставлена ли дискета в дисковод, надо воспользоваться свойством IsReady объекта Drive. Это свойство возвращает значение True, если специфицированный диск доступен, и значение False, если не доступен (листинг 13.2).

Листинг 13.2. Проверка доступности диска

```
Sub AccessableDrive()
   Dim fs As New FileSystemObject
   Dim dn  As String
```

```
   dn = "a:"
   Dim d As Drive
   Set d = fs.GetDrive(dn)
   If d.IsReady Then
      MsgBox "Диск доступен"
   Else
      MsgBox "Диск не доступен. Вставьте дискету"
   End If
End Sub
```

Получение информации о диске

Свойства объекта `Drive` позволяют собрать разнообразную информацию о диске, начиная от его размера и заканчивая используемой файловой системой. Следующий код (листинг 13.3) демонстрирует подобное применение объекта `Drive`. В результате использования этой программы на компьютере автора данной книги отобразилось сообщение, показанное на рис. 13.1.



Рис. 13.1. Получение информации о диске

Листинг 13.3. Получение информации о диске

```
Sub DriveInfo()
   Dim fs As New FileSystemObject
   Dim d  As Drive
   Dim drvPath As String
   drvPath = "c:\"
   If fs.DriveExists(drvPath) Then
      Dim s As String
      Dim t As String
      Set d = fs.GetDrive(fs.GetDriveName(drvPath))
      s = "Drive " & UCase(drvPath) & " - "
      s = s & d.VolumeName & vbCr
      s = s & "Free Space: " & FormatNumber(d.FreeSpace / 1024, 0) _
        & " Kbytes" & vbCr
```

```
      s = s _
        & "Avaiable Space: " & FormatNumber(d.AvailableSpace / 1024, 0) _
        & " Kbytes" & vbCr
      s = s & "Avaiable Space: " & FormatNumber(d.TotalSize / 1024, 0) _
        & " Kbytes" & vbCr
      s = s & "File System: " & d.FileSystem & vbCr
      Select Case d.DriveType
         Case 0: t = "Unknown"
         Case 1: t = "Removable"
         Case 2: t = "Fixed"
         Case 3: t = "Network"
         Case 4: t = "CD-ROM"
         Case 5: t = "RAM Disk"
      End Select
      s = s & "Drive Type: " & t
      MsgBox s
   End If
End Sub
```

Объект *Folder*

Объект `Folder` инкапсулирует в себе информацию о каталоге. Методы объекта, перечисленные в табл. 13.4, позволяют создавать файл, копировать, перемещать и удалять каталог. Свойства этого объекта, перечисленные в табл. 13.5, включают большой объем информации о каталоге от времени его создания до списка всех его файлов и подкаталогов.

Таблица 13.4. *Методы объекта* `Folder`

Метод	Описание
`Copy`	Копирует каталог или группу каталогов
`CreateTextFile`	Открывает текстовый файл
`Delete`	Удаляет каталог
`Move`	Перемещает каталог или группу каталогов

Таблица 13.5. *Свойства объекта* `Folder`

Свойство	Описание
`Attributes`	Возвращает атрибуты каталога
`DateCreated`	Возвращает дату и время создания каталога

Таблица 13.5 *(окончание)*

Свойство	Описание
`DateLastAccessed`	Возвращает дату и время последнего обращения к каталогу
`DateLastModified`	Возвращает дату и время последней модификации каталога
`Drive`	Возвращает имя дисковода, в котором расположен данный каталог
`Files`	Возвращает семейство `Files` файлов данного каталога
`IsRootFolder`	Проверяет, является ли каталог корневым
`Name`	Имя каталога
`ParentFolder`	Возвращает родительский каталог
`Path`	Возвращает путь к каталогу
`ShortName`	Возвращает короткое имя каталога
`Size`	Суммарный размер в байтах всех файлов и подкаталогов данного каталога
`SubFolders`	Возвращает семейство `Folders` подкаталогов данного каталога
`Type`	Возвращает тип каталога

Как проверить, существует ли каталог

Для того чтобы проверить, существует ли каталог, надо воспользоваться методом `FolderExists` объекта `FileSystemObject`. Этот метод возвращает значение `True`, если специфицированный каталог существует, и значение `False`, если не существует (листинг 13.4).

Листинг 13.4. Проверка существования каталога

```
Sub CheckFolder()
   Dim fs As New FileSystemObject
   Dim fn  As String
   fn = "c:\windows"
   If fs.FolderExists(fn) Then
      MsgBox fn & " существует"
   Else
      MsgBox fn & " не существует"
   End If
End Sub
```

Создание каталога

Каталог создается методом `FolderExists` объекта `FileSystemObject`.

```
CreateFolder foldername
```

Здесь *foldername* — имя создаваемого каталога. Следующий код (листинг 13.5) демонстрирует применение этого метода, причем, прежде чем конструировать каталог, проверяется, существует ли он.

Листинг 13.5. Создание каталога

```
Sub Create()
   Dim fs As New FileSystemObject
   Dim fl As Folder
   Dim fn As String
   fn = "c:\myfolder"
   If Not fs.FolderExists(fn) Then
      Set fl = fs.CreateFolder(fn)
   End If
End Sub
```

Копирование каталога

Копирование каталога производится методом `CopyFolder` объекта `FileSystemObject` либо методом `Copy` объекта `Folder`.

```
CopyFolder source, destination[,overwritefiles]
```

- *source* — обязательный параметр, задающий имя копируемого каталога;
- *destination* — обязательный параметр, указывающий имя целевого каталога, в который копируется исходный каталог;
- *overwritefiles* — необязательный параметр, определяющий, надо ли при копировании переписывать целевой каталог, если он уже существует. При этом, если целевой каталог уже существует, при значении этого параметра, равном `False`, генерируется ошибка.

Например, в следующем коде (листинг 13.6) производится копирование каталога C:\firstfolder в каталог C:\myfolder с переписыванием его содержания, если такой каталог уже существует.

Листинг 13.6. Копирование каталога методом `CopyFolder`

```
Sub DemoCopyFolder()
   Dim fs As New FileSystemObject
```

```
   fs.CopyFolder Source:="c:\firstfolder", Destination:="c:\myfolder", _
                 Overwritefiles:=True
End Sub
```

Метод `Copy` объекта `Folder` также копирует каталог, причем он действует аналогично методу `CopyFolder`.

```
Copy destination[,overwritefiles]
```

Следующий код (листинг 13.7) демонстрирует применение метода `Copy` при копировании файлов. При этом производится как проверка существования каталога источника, так и запрос пользователя о подтверждении перезаписи целевого каталога, если таковой уже существует.

Листинг 13.7. Копирование каталога методом `Copy`

```
Sub DemoCopy()
   Dim fs As New FileSystemObject
   Dim flsrc As Folder
   Dim fnsrc As String
   Dim fndest As String
   fnsrc = "c:\firstfolder "
   fndest = "c:\myfolder"
   If fs.FolderExists(fnsrc) Then
      Set flsrc = fs.GetFolder(fnsrc)
      If Not fs.FolderExists(fndest) Then
          flsrc.Copy Destination:=fndest, overwritefiles:=False
      ElseIf MsgBox("Overwite", vbQuestion + vbYesNo) = vbYes Then
          flsrc.Copy Destination:=fndest, overwritefiles:=True
      End If
   End If
End Sub
```

Перемещение каталога

Перемещение каталога производится методом `MoveFolder` объекта `FileSystemObject` либо методом `Move` объекта `Folder`.

```
MoveFolder source, destination
```

- *`source`* — обязательный параметр, задающий имя перемещаемого каталога;
- *`destination`* — обязательный параметр, указывающий имя целевого каталога, в который перемещается исходный каталог. Если целевой каталог уже существует, то генерируется ошибка.

Метод `Move` объекта `Folder` действует аналогично методу `CopyFolder`.

```
Move destination
```

Например, в следующем коде (листинг 13.8) перемещается каталог c:\firstfolder в c:\myfolder. При этом во избежание генерации ошибки проверяется наличие обоих каталогов.

Листинг 13.8. Перемещение каталога

```
Sub DemoMove()
   Dim fs As New FileSystemObject
   Dim fnsrc As String
   Dim fndest As String
   fnsrc = "c:\firstfolder"
   fndest = "c:\myfolder"
   If fs.FolderExists(fnsrc) And Not fs.FolderExists(fndest) Then
      fs.MoveFolder fnsrc, fndest
      MsgBox "Перемещен"
  Else
      MsgBox "Перемещение не допустимо"
  End If
End Sub
```

Удаление каталога

Удаление каталога производится методом `DeleteFolder` объекта `FileSystemObject` и методом `Delete` объекта `Folder`.

```
DeleteFolder folderspec[, force]
```

- *folderspec* — обязательный параметр, задающий имя удаляемого каталога;
- *force* — необязательный параметр, определяющий, надо ли удалять каталоги "только для чтения". При значении этого параметра, равном `False`, при попытке удаления каталога "только для чтения" генерируется ошибка.

Метод `Delete` объекта `Folder` действует аналогично методу `DeleteFolder`.

```
Delete force
```

Следующий код (листинг 13.9) демонстрирует, как обработать ошибку, которая будет сгенерирована при попытке удаления каталога "только для чтения".

Листинг 13.9. Удаление каталога

```
Sub DemoDelete()
   Dim fs As New FileSystemObject
```

```
   Dim fn As String
   fn = "c:\myfolder"
   On Error GoTo errorhandler
   If fs.FolderExists(fn) Then
      fs.DeleteFolder fn, False
      MsgBox "Файл удален"
   End If
   Exit Sub
errorhandler:
   MsgBox "Permission Denied"
End Sub
```

Получение информации о каталоге

Свойства объекта `Folder` позволяют собрать разнообразную информацию о каталоге, его атрибутах. Следующий код (листинг 13.10) демонстрирует подобное применение объекта `Folder`. В результате использования этой программы на компьютере автора отобразилось сообщение, показанное на рис. 13.2.

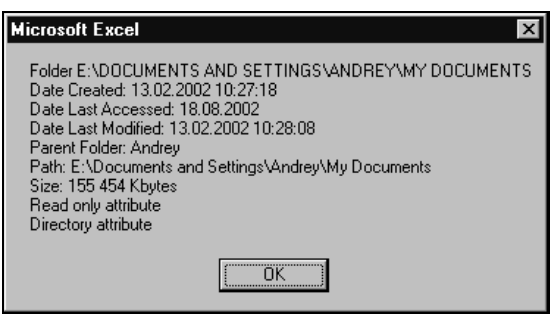


Рис. 13.2. Получение информации о каталоге

Листинг 13.10. Получение информации о каталоге

```
Sub FolderInfo()
   Dim fs As New FileSystemObject
   Dim fn As String
   fn = CurDir
   If fs.FolderExists(fn) Then
      Dim s As String
      Dim fl  As Folder
      Set fl = fs.GetFolder(fn)
      s = "Folder " & UCase(fn) & vbCr
      s = s & "Date Created: " & fl.DateCreated & vbCr
```

```
      s = s & "Date Last Accessed: " & fl.DateLastAccessed & vbCr
      s = s & "Date Last Modified: " & fl.DateLastModified & vbCr
      s = s & "Parent Folder: " & fl.ParentFolder.Name & vbCr
      s = s & "Path: " & fl.Path & vbCr
      s = s & "Size: " & FormatNumber(fl.Size / 1024, 0) _
              & " Kbytes" & vbCr
      If fl.Attributes And 0 Then s = s & "Normal attribute" & vbCr
      If fl.Attributes And 1 Then s = s & "Read only attribute" & vbCr
      If fl.Attributes And 2 Then s = s & "Hidden attribute" & vbCr
      If fl.Attributes And 4 Then s = s & "System attribute" & vbCr
      If fl.Attributes And 16 Then s = s & "Directory attribute" & vbCr
      If fl.Attributes And 32 Then s = s & "Archive attribute" & vbCr
      If fl.Attributes And 128 Then s = s & "Compressed attribute" & vbCr
      MsgBox s
   End If
End Sub
```

Определение уровня данного каталога

Свойство `IsRootFolder` определяет, является ли данный каталог корневым, а свойство `ParentFolder` возвращает родительский каталог. Использование этих свойств позволяет последовательно перебираться по восходящей от текущего к родительскому каталогу до тех пор, пока не будет достигнут корневой каталог. Приведенный алгоритм (листинг 13.11) позволяет найти уровень данного каталога.

Листинг 13.11. Определение уровня данного каталога

```
Sub DemoLevelCounter()
   Dim fs As New FileSystemObject
   Dim fl As Folder
   Dim n As Integer
   n = 0
   Set fl = fs.GetFolder(CurDir)
   If fl.IsRootFolder Then
      MsgBox "Корневой каталог"
   Else
      Do Until fl.IsRootFolder
         Set fl = fl.ParentFolder
         n = n + 1
      Loop
      MsgBox "Каталог " & CurDir & Space(1) & n & "-го уровня"
   End If
End Sub
```

Список подкаталогов данного каталога

Свойство `SubFolders` объекта `Folder` возвращает семейство `Folders` всех подкаталогов данного каталога. Поэтому, если требуется получить список подкаталогов данного каталога, достаточно воспользоваться этим свойством (листинг 13.12).

Листинг 13.12. Список подкаталогов данного каталога

```
Sub ListSubFolders()
   Dim fs As New FileSystemObject
   Dim fl As Folder
   Dim fld As Folder
   Dim flds As Folders
   Dim s As String
   Set fl = fs.GetFolder("c:\")
   Set flds = fl.SubFolders
   For Each fld In flds
        s = s & fld.Name & vbCr
   Next
   MsgBox s
End Sub
```

Повторив данную процедуру рекурсивно, можно определить список всех подкаталогов данного каталога (листинг 13.13).

Листинг 13.13. Список всех подкаталогов данного каталога

```
Private fs As New FileSystemObject
Private i As Long

Sub StartFolders()
   Dim fl As Folder
   Set fl = fs.GetFolder("c:\JBuilder6")
   i = 1
   ListSubFolders fl
End Sub

Sub ListSubFolders(ByVal fl As Folder)
   Dim sf As Folders
   Set sf = fl.SubFolders
   If sf.Count > 0 Then
      For Each fld In sf
          ListSubFolders fld
```

```
            Cells(i, 1).Value = fld.Path
            i = i + 1
        Next
    End If
End Sub
```

Объект *File*

Объект `File` инкапсулирует в себе данные и методы по работе с файлом. Эти методы и свойства перечислены в табл. 13.6 и 13.7 и позволяют производить широкий спектр действий над файлами от определения или изменения их атрибутов до перемещения и удаления файлов.

Таблица 13.6. *Методы объекта* `File`

Метод	Описание
`Copy`	Копирует файл
`Delete`	Удаляет файл
`Move`	Перемещает файл
`OpenAsTextStream`	Создает объект `TextStream`

Таблица 13.7. *Свойства объекта* `File`

Свойство	Описание
`Attributes`	Атрибуты файла
`DateCreated`	Дата и время создания файла
`DateLastAccessed`	Дата и время последнего доступа к файлу
`DateLastModified`	Дата и время последней модификации файла
`Drive`	Имя диска, на котором расположен файл
`Name`	Имя файла
`ParentFolder`	Родительский каталог
`Path`	Путь к файлу
`ShortName`	Короткое имя файла
`ShortPath`	Короткий путь к файлу
`Size`	Размер файла
`Type`	Тип файла

Как проверить, существует ли файл

Для того чтобы проверить, существует ли файл, надо воспользоваться методом `FileExists` объекта `FileSystemObject`. Этот метод возвращает значение `True`, если специфицированный файл существует, и значение `False`, если не существует (листинг 13.14).

Листинг 13.14. Проверка существования файла

```
Sub CheckFile()
   Dim fs As New FileSystemObject
   Dim fn  As String
   fn = "c:\temp.txt"
   If fs.FileExists(fn) Then
      MsgBox fn & " существует"
   Else
      MsgBox fn & " не существует"
   End If
End Sub
```

Копирование и перемещение файла

Копирование файла производится методом `CopyFile` объекта `FileSystemObject` либо методом `Copy` объекта `File`.

```
CopyFile source, destination[, overwritefiles]
```

- *source* — обязательный параметр, задающий имя копируемого файла;
- *destination* — обязательный параметр, указывающий имя целевого файла, в который копируется исходный файл;
- *overwritefiles* — необязательный параметр, определяющий, надо ли при копировании переписывать целевой файл, если он уже существует. При этом, если целевой файл уже существует, при значении этого параметра, равном `False`, генерируется ошибка.

Метод `Copy` объекта `File` действует аналогично методу `CopyFile`.

```
Copy destination[, overwrite]
```

Например, в следующем коде (листинг 13.15) производится копирование файла C:\Temp.txt в C:\TempNew.txt в случае, если последнего не существует.

Листинг 13.15. Копирование файла

```
Sub DemoCopyFile()
   Dim fs As New FileSystemObject
```

```
   Dim fsrc  As String
   Dim fdst  As String
   fsrc = "c:\temp.txt"
   fdst = "c:\tempNew.txt"
   On Error GoTo errorhandler
   If fs.FileExists(fsrc) Then
      fs.CopyFile fsrc, fdst, False
   End If
   Exit Sub
errorhandler:
   MsgBox fdst & " exists"
End Sub
```

Перемещение файла осуществляется методом `MoveFile` объекта `FileSystemObject` и методом `Move` объекта `File`.

```
MoveFolder source, destination
```

- *source* — обязательный параметр, задающий имя перемещаемого файла;
- *destination* — обязательный параметр, указывающий имя целевого файла, в который перемещается исходный файл. Если целевой файл уже существует, то генерируется ошибка.

Метод `Move` объекта `File` действует аналогично методу `MoveFile`.

```
Move destination
```

Удаление файла

Удаление файла производится методом `DeleteFile` объекта `FileSystemObject` и методом `Delete` объекта `File`.

```
DeleteFile filespec[, force]
```

- *filespec* — обязательный параметр, задающий имя удаляемого файла;
- *force* — необязательный параметр, определяющий, надо ли удалять файлы "только для чтения". При значении этого параметра, равном `False`, при попытке удаления каталога "только для чтения" генерируется ошибка.

Метод `Delete` объекта `File` действует аналогично методу `DeleteFile`.

```
Delete force
```

Следующий код (листинг 13.16) демонстрирует то, как удаляется файл с предварительной проверкой его существования.

Листинг 13.16. Удаление файла

```
Sub DemoDeleteFile()
   Dim fs As New FileSystemObject
```

```
   Dim f As String
   f = "c:\temp.txt"
   If fs.FileExists(f) Then
      fs.DeleteFile f
   End If
End Sub
```

Информация о файле

Свойства объекта `File` позволяют собрать разнообразную информацию о файле, его местоположении и атрибутах. Следующий код (листинг 13.17) демонстрирует подобное применение объекта `File` для определения параметров рабочей книги с выполняемым кодом (рис. 13.3).

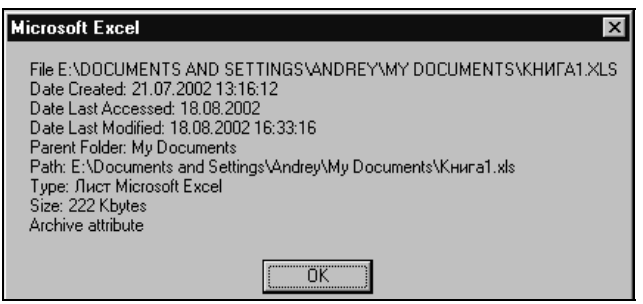


Рис. 13.3. Получение информации о рабочей книге

Листинг 13.17. Получение информации о файле

```
Sub FileInfo()
   Dim fs As New FileSystemObject
   Dim fn As String
   fn = ThisWorkbook.FullName
   If fs.FileExists(fn) Then
      Dim s As String
      Dim f  As File
      Set f = fs.GetFile(fn)
      s = "File " & UCase(fn) & vbCr
      s = s & "Date Created: " & f.DateCreated & vbCr
      s = s & "Date Last Accessed: " & f.DateLastAccessed & vbCr
      s = s & "Date Last Modified: " & f.DateLastModified & vbCr
      s = s & "Parent Folder: " & f.ParentFolder.Name & vbCr
      s = s & "Path: " & f.Path & vbCr
      s = s & "Type: " & f.Type & vbCr
      s = s & "Size: " & FormatNumber(f.Size / 1024, 0) _
              & " Kbytes" & vbCr
```

```
      If f.Attributes And 0 Then s = s & "Normal attribute" & vbCr
      If f.Attributes And 1 Then s = s & "Read only attribute" & vbCr
      If f.Attributes And 2 Then s = s & "Hidden attribute" & vbCr
      If f.Attributes And 4 Then s = s & "System attribute" & vbCr
      If f.Attributes And 16 Then s = s & "Directory attribute" & vbCr
      If f.Attributes And 32 Then s = s & "Archive attribute" & vbCr
      If f.Attributes And 128 Then s = s & "Compressed attribute" & vbCr
      MsgBox s
   End If
End Sub
```

Изменение атрибутов файла

Атрибуты файлов при помощи свойства `Attributes` можно не только считывать, но и устанавливать и удалять. Следующий код (листинг 13.18) демонстрирует, как это можно делать на примере атрибута `archive`. Сначала проверяется, установлен ли такой атрибут у некоторого специфицированного файла (в данном случае C:\Tmp.txt). Если он установлен, то пользователю предлагается либо удалить его, либо оставить не тронутым. Если же атрибут не установлен, то пользователю предоставляется возможность его установить. Для идентификации атрибутов удобно использовать константы, перечисленные в табл. 13.8.

Таблица 13.8. *Константы атрибутов файла*

Константа	Значение	Описание
vbNormal	0	Обычный
vbReadOnly	1	Только чтение
vbHidden	2	Скрытый
vbSystem	4	Системный
vbDirectory	16	Каталог или папка
vbArchive	32	Файл был изменен после последнего резервирования

Листинг 13.18. Изменение атрибутов файла

```
Sub DemoArchiveAttribute()
   Dim fs As New FileSystemObject
   Dim f As File
   Dim filespec As String
   filespec = "c:\tmp.txt"
```

```
   If fs.FileExists(filespec) Then
      Set f = fs.GetFile(filespec)
      If f.Attributes And vbArchive Then
         If MsgBox("Удалить атрибут archive?", vbYesNo) = vbYes Then
            f.Attributes = f.Attributes – vbArchive
            MsgBox "Атрибут archive удален"
         End If
      Else
         If MsgBox("Установить атрибут Archive?", vbYesNo) = vbYes Then
            f.Attributes = f.Attributes + vbArchive
            MsgBox "Атрибут archive установлен"
         End If
      End If
   End If
End Sub
```

Список всех файлов данного каталога

Свойство `Files` объекта `Folder` возвращает семейство `Files` всех файлов данного каталога. Поэтому, если требуется получить список файлов данного каталога, достаточно воспользоваться этим свойством (листинг 13.19).

Листинг 13.19. Список файлов данного каталога

```
Sub ListFiles()
   Dim fs As New FileSystemObject
   Dim fl As Folder
   Dim fls As Files
   Dim f As File
   Dim s As String
   Set fl = fs.GetFolder("c:\")
   Set fls = fl.Files
   For Each f In fls
        s = s & f.Name & vbCr
   Next
   MsgBox s
End Sub
```

Объект *TextStream*

Объект `TextStream` инкапсулирует себе свойства и методы доступа к файлу, обеспечивающие возможность считывать, записывать и добавлять данные в файл. В табл. 13.9 и 13.10 перечислены методы и свойства этого объекта.

Таблица 13.9. *Методы объекта* `TextStream`

Метод	Описание
`Close`	Закрытие файла
`Read`	Считывает указанное число символов
`ReadAll`	Считывает все содержание файла
`ReadLine`	Считывает целую строку из файла
`Skip`	Пропускает указанное число символов при считывании данных из файла
`SkipLine`	Пропускает строку при считывании данных из файла
`Write`	Записывает в файл указанную строку текста
`WriteBlankLines`	Записывает в файл указанное число пустых строк, заканчивающихся символом перехода на новую строку
`WriteLine`	Записывает в файл указанную строку текста, заканчивающуюся символом перехода на новую строку

Таблица 13.10. *Свойства объекта* `TextStream`

Свойство	Описание
`AtEndOfLine`	Возвращает значение `True`, если указатель находится в конце строки
`AtEndOfStream`	Возвращает значение `True`, если указатель находится в конце файла
`Column`	Текущая колонка указателя
`Line`	Текущая строка указателя

Открытие файла и создание объекта *TextStream*

Для открытия файла и создания объекта `TextStream` имеются метод `OpenAsTextStream` объекта `File` и два метода `CreateTextFile` и `OpenTextStream` объекта `FileSystemObject`, причем наиболее универсальным из этих методов является последний.

Метод *OpenAsTextStream*

Метод `OpenAsTextStream` объекта `File` открывает указанный файл и создает объект `TextStream`, обеспечивающий специфицированный доступ, а имен-

но — для чтения, добавления или считывания данных из файла. При попытке открыть несуществующий файл метод генерирует ошибку.

```
OpenAsTextStream([iomode, [format]])
```

- *iomode* — необязательный параметр, задающий то, для какой операции открывается файл. Допустимыми значениями являются следующие константы: `ForReading`, `ForWriting` и `ForAppending`;
- *format* — необязательный параметр, задающий тип файла. Допустимыми значениями являются следующие константы: `TristateUseDefault` (тип файла, используемый в системе по умолчанию), `TristateTrue` (Unicode-файл), `TristateFalse` (ASCII-файл).

Примечание

VBA использует кодировку Unicode для хранения строк и манипулирования ими. Unicode — это схема кодирования символов, использующая два байта для представления *каждого* символа. Международная организация стандартов (International Standards Organization, ISO) определяет число из диапазона от 0 до 65 535 (2^{16} — 1) для каждого символа из каждого языка (плюс некоторое количество чисел для будущего использования). Во всех 32-разрядных версиях Windows стандарт Unicode используется *компонентной моделью объектов* (Component Object Model, COM), основной для технологий OLE и ActiveX.

В следующем коде (листинг 13.20) открывается существующий файл C:\Test.txt. Создается объект `TextStream` для записи в него строки "Hello, World!", после чего этот объект закрывается. Затем еще раз создается объект `TextStream`, теперь для добавления в него строки "Hello, World!", после чего этот объект закрывается. И наконец, в третий раз создается объект `TextStream`, но теперь для считывания содержимого файла.

Листинг 13.20. Запись, добавление и считывание данных

```
Sub DemoOpenAsTextStream()
   Dim fs As New FileSystemObject
   Dim f As File
   Dim fn As String
   fn = "c:\test.txt"
   If fs.FileExists(fn) Then
      Set f = fs.GetFile(fn)
      Dim ts As TextStream
      Set ts = f.OpenAsTextStream(ForWriting) ' Запись данных
      ts.WriteLine "Hello, World!"
      ts.Close
      Set ts = f.OpenAsTextStream(ForAppending) ' Добавление данных
      ts.WriteLine "Hello, again!"
```

```
      ts.Close
      Set ts = f.OpenAsTextStream(ForReading) ' Считывание данных
      Dim msg As String
      msg = ts.ReadAll
      MsgBox msg
      ts.Close
   Else
      MsgBox "No file"
   End If
End Sub
```

Метод *CreateTextFile*

Метод `CreateTextFile` объекта `FileSystemObject` создает файл и объект `TextStream`, обеспечивающий доступ к данным файла. В этом методе не специфицируется операция, для которой открыт файл.

```
CreateTextFile(filename[, overwrite[, unicode]])
```

- *filename* — обязательный параметр, специфицирующий имя создаваемого файла;
- *overwrite* — необязательный параметр, определяющий, надо ли перезаписывать файл, если он уже существует. Если значение равно `True`, то перезаписывается, если `False`, то не перезаписывается. При существовании файла на диске и значении параметра, равном `False`, метод генерирует ошибку;
- *unicode* — определяет код создаваемого текстового файла. Если значение параметра равно `True`, то файл создается как Unicode-файл. Если значение параметра равно `False`, то файл создается как ASCII-файл.

В следующем коде создается файл C:\Testfile.txt для записи в него строки "Hello, World!". Так как при существовании файла на диске и значении параметра *overwrite*, равном `False`, метод `CreateTextFile` генерирует ошибку, то при использовании этого метода можно заранее не проверять существование файла, а ограничиться обработкой возможной ошибки.

Листинг 13.21. Создание файла и объекта `TextStream` методом `CreateTextFile`

```
Sub DemoCreateTextFile()
   Dim fs As New FileSystemObject
   Dim ts As TextStream
   On Error GoTo errorhandler
   Set ts = fs.CreateTextFile("c:\testfile.txt", False)
   ts.WriteLine "Hello, World!"
```

```
   ts.Close
   Exit Sub
errorhandler:
   MsgBox "File already exists"
End Sub
```

Метод *OpenTextFile*

Метод `OpenTextFile` объекта `FileSystemObject` открывает файл и создает объект `TextStream`, обеспечивающий доступ к файлу.

```
OpenTextFile(filename[, iomode[, create[, format]]])
```

- *filename* — обязательный параметр, идентифицирующий имя открываемого файла;
- *iomode* — необязательный параметр, задающий, для какой операции открывается файл. Допустимыми значениями являются следующие константы: `ForReading`, `ForWriting` и `ForAppending`;
- *create* — необязательный параметр, который определяет, надо ли создавать новый файл, специфицированный параметром *filename*, если он не существует;
- *format* — необязательный параметр, задающий тип файла. Допустимыми значениями являются следующие константы: `TristateUseDefault` (тип файла, используемый в системе по умолчанию), `TristateTrue` (Unicode-файл), `TristateFalse` (ASCII-файл).

Например, в следующем коде (листинг 13.22) открывается файл C:\Testfile.txt для записи данных и создается объект `TextStream` для доступа к его данным, причем если такового файла не существует, то он создается. В файл вводится всего одна строка "Hello, World!", после чего происходит закрытие объекта `TextStream`.

Листинг 13.22. Открытие файла и создание объекта `TextStream` методом `OpenTextFile`

```
Sub DemoOpenTextFile()
   Dim fs As New FileSystemObject
   Dim ts As TextStream
   Set ts = fs.OpenTextFile("c:\testfile.txt", ForAppending, True)
   ts.WriteLine "Hello, World!"
   ts.Close
End Sub
```

Ввод и вывод данных из поля ввода в текстовый файл

В рассматриваемом в этом разделе примере демонстрируется, как из текстового файла можно ввести данные в поле ввода, и наоборот, из поля сохранить данные в текстовый файл. Для реализации примера создайте форму, на которой расположите поле ввода и две кнопки. В модуле формы наберите приводимый ниже код (листинг 13.23). При нажатии кнопки **Считывание** данные из файла C:\Test.txt заносятся в поле ввода, а при нажатии кнопки **Сохранение** данные из поля ввода сохраняются в упомянутом файле. Отметим, что в коде также осуществляется проверка на пустоту специфицированного файла. Если файл пуст, то в поле ввода он не загружается, а просто осуществляется очистка поля ввода. Для корректной работы программы не забудьте установить ссылку на Microsoft Scripting Runtime в диалоговом окне **References**, которое отображается на экране выбором команды **Project | References**.

Листинг 13.23. Ввод данных из текстового файла в поле ввода

```
Private fname As String

Private Sub UserForm_Initialize()
   fname = "c:\test.txt"
   TextBox1.MultiLine = True
   Me.Caption = "Считывание и сохранение "
   CommandButton1.Caption = "Считывание"
   CommandButton2.Caption = "Сохранение"
End Sub

Private Sub CommandButton1_Click()
   Dim fs As New FileSystemObject
   If fs.FileExists(fname) Then
      Dim ts As TextStream
      Set ts = fs.OpenTextFile(fname, ForReading)
      If ts.AtEndOfStream Then ' Проверяется, пуст ли файл
         TextBox1.Text = Empty
      Else
         TextBox1.Text = ts.ReadAll
      End If
      ts.Close
    Else
       MsgBox "No file " & fname
    End If
End Sub
```

```
Private Sub CommandButton2_Click()
   Dim fs As New FileSystemObject
   Dim ts As TextStream
   Set ts = fs.OpenTextFile(fname, ForWriting, True)
   ts.Write (TextBox1.Text)
   ts.Close
End Sub
```

Запись, присоединение и считывание данных из файла

В качестве примера записи, присоединения и считывания данных из файла приведем программу (листинг 13.24), которая создает таблицу табуляции функции Sin(*i*) при *i*, изменяющемся от 1 до 19. При этом сначала создается файл C:\Table.txt, если его еще не было, для того чтобы ввести результат табуляции при изменении *i* от 1 до 9, затем файл повторно открывается, для того чтобы присоединить оставшиеся значения, и в заключение файл еще раз открывается, но уже для считывания данных. Для корректной работы программы не забудьте установить ссылку на Microsoft Scripting Runtime в диалоговом окне **References**, которое отображается на экране выбором команды **Project | References**.

Листинг 13.24. Создание таблицы умножения в файле

```
Sub DemoOperations()
   Dim fs As New FileSystemObject
   Dim ts As TextStream
   Dim i As Integer, j As Integer
   Set ts = fs.OpenTextFile("C:\Table.txt", ForWriting, True)
   For i = 1 To 9
      ts.Write (i)
      ts.Write (vbTab)
      ts.Write (FormatNumber(Sin(i), 3))
      ts.WriteLine
   Next
   ts.Close

   Set ts = fs.OpenTextFile("C:\Table.txt", ForAppending, False)
   For i = 10 To 19
      ts.WriteLine (i & vbTab & FormatNumber(Sin(i), 3))
   Next
   ts.Close

   Set ts = fs.OpenTextFile("C:\Table.txt", ForReading, False)
   Do While Not ts.AtEndOfStream
       Debug.Print ts.ReadLine
```

```
    Loop
    ts.Close
End Sub
```

Запись данных в HTML-файл

Используя объект `TextStream`, данные можно записывать не только в текстовый, но в HTML-файл, если вспомнить, что последний является обычным текстовым файлом с управляющими тегами. В качестве примера создадим приложение (листинг 13.25), которое создает HTML-отчет о результатах бросаний игральной кости. Итак, создайте форму, на которой расположите кнопку и поле ввода. В поле будет вводиться число бросков. При нажатии кнопки **Отчет** автоматически будет генерироваться файл C:\Report.htm, в котором содержится таблица результатов игры (рис. 13.4). Для корректной работы программы не забудьте установить ссылку на Microsoft Scripting Runtime в диалоговом окне **References**, которое отображается на экране выбором команды **Project | References**.

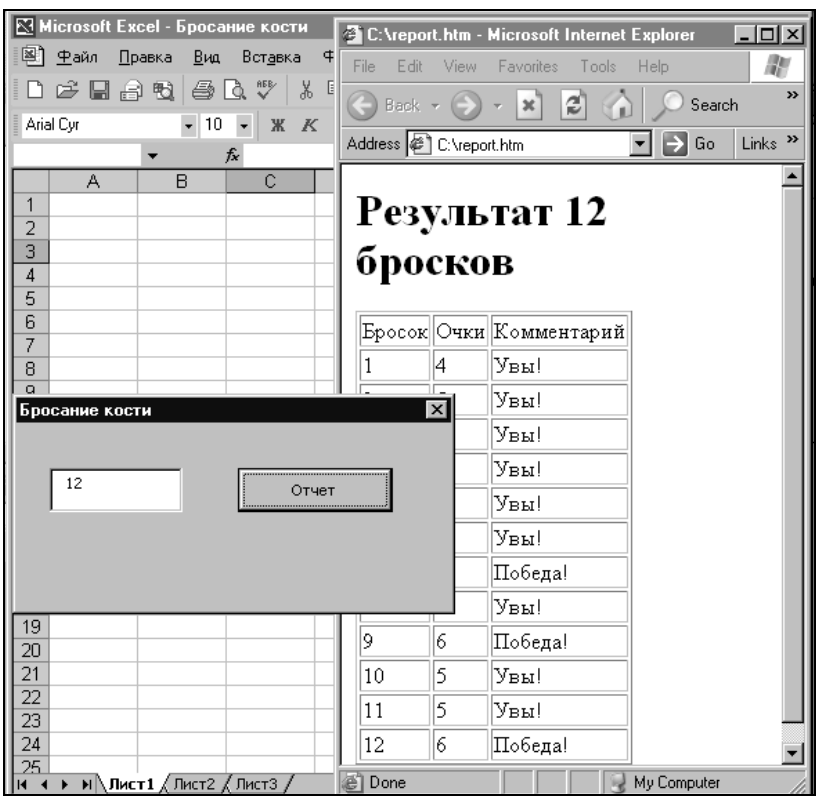


Рис. 13.4. Запись данных в HTML-файл

Листинг 13.25. Запись данных в HTML-файл

```
Private Sub UserForm_Initialize()
   CommandButton1.Caption = "Отчет"
   Me.Caption = "Бросание кости"
End Sub

Private Sub CommandButton1_Click()
   Dim fs As New FileSystemObject
   Dim ts As TextStream
   Set ts = fs.OpenTextFile("c:\report.htm", ForWriting, True)
   Dim n As Integer
   Dim i As Integer
   Dim r As Integer
   Dim s As String
   n = CInt(TextBox1.Text)
   Randomize
   ts.WriteLine ("<html>")
   ts.WriteLine ("<body>")
   ts.WriteLine ("<h1> Результат " & n & " бросков</h1>")
   ts.WriteLine ("<table border=1 width=50%>")
   ts.WriteLine ("<tr>")
   ts.WriteLine ("<td>Бросок</td>")
   ts.WriteLine ("<td>Очки</td>")
   ts.WriteLine ("<td>Комментарий</td>")
   ts.WriteLine ("</tr>")
   For i = 1 To n
     ts.WriteLine ("<tr>")
     ts.WriteLine ("<td>" & i & "</td>")
     r = 5 * Rnd() + 1
     ts.WriteLine ("<td>" & r & "</td>")
     If r = 6 Then
        s = "Победа!"
      Else
         s = "Увы!"
      End If
      ts.WriteLine ("<td>" & s & "</td>")
      ts.WriteLine ("</tr>")
   Next
   ts.WriteLine ("</body>")
   ts.WriteLine ("</html>")
   ts.Close
End Sub
```

Объект *Dictionary*

Объект `Dictionary` *(словарь)* позволяет хранить динамический набор объектов любой природы, идентифицируемых по ключу. В качестве ключа обычно используется либо целое число, либо буква, либо набор букв. Методы и свойства этого объекта, перечисленные в табл. 13.11 и 13.12, позволяют добавлять/удалять элементы в словарь в любом порядке и находить нужный элемент.

Таблица 13.11. *Методы объекта* `Dictionary`

Метод	Описание
`Add`	Добавляет элемент в словарь
`Exists`	Проверяет наличие ключа в словаре
`Items`	Массив всех элементов словаря
`Keys`	Массив всех ключей словаря
`Remove`	Удаляет элемент из словаря
`RemoveAll`	Удаляет все элементы из словаря

Таблица 13.12. *Свойства объекта* `Dictionary`

Свойство	Описание
`CompareMode`	Устанавливает способ сравнения ключей. Допустимыми значениями являются следующие константы: `vbUseCompareOption`, `vbBinaryCompare` и `vbTextCompare`
`Count`	Число элементов словаря
`Item`	Элемент словаря
`Key`	Ключ элемента словаря

Создание адресной книги с помощью объекта *Dictionary*

В качестве примера создадим объект `Dictionary`, содержащий в себе адресную книгу. Для этого нам потребуется построить модуль класса `Firm` с полями `FirmName`, `Location`, `Website`, в которых и будет содержаться информация, учитываемая в адресной книге. Далее все очень просто. Сначала в коде

создаются два экземпляра класса Firm для ввода новых данных в адресную книгу. Затем все ее элементы выводятся в окно **Immediate**. После чего по ключу удаляется один элемент книги, а содержание обновленной книги опять выводится в окно **Immediate**. Прежде чем удалять элемент из книги, проверяется наличие соответствующего ключа в словаре. Для корректной работы программы (листинг 13.26) не забудьте установить ссылку на Microsoft Scripting Runtime в диалоговом окне **References**, которое отображается на экране выбором команды **Project | References**.

Листинг 13.26, *а*. Словарь

```
Sub DemoDictionary()
   Dim d As New Dictionary
   Dim f1 As New Firm
   With f1
      .FirmName = "BHV"
      .Location = "Russia"
      .Website = "www.bhv.ru"
   End With
   d.Add "bhv", f1
   Debug.Print (f1.FirmName)

   Dim f2 As New Firm
   With f2
      .FirmName = "Microsoft"
      .Location = "US"
      .Website = "www.microsoft.com"
   End With
   d.Add "ms", f2
   Debug.Print f2.FirmName

   If d.Exists("bhv") Then
      Debug.Print d.Item("bhv").Location
   End If

   Debug.Print
   Dim f As Firm
   Dim i As Integer
   Dim s As String
   Dim a As Variant
   a = d.Items
   For i = 0 To d.Count - 1
      Set f = a(i)
```

```
      With f
         Debug.Print .FirmName & vbTab & .Location & vbTab & .Website
      End With
   Next

   Debug.Print
   If d.Exists("ms") Then
     d.Remove ("ms")
   End If

   a = d.Items
   For i = 0 To d.Count - 1
      Set f = a(i)
      With f
         Debug.Print .FirmName & vbTab & .Location & vbTab & .Website
      End With
   Next
End Sub
```

Листинг 13.26, *б*. Словарь. Модуль класса `Firm`

```
Public FirmName As String
Public Location As String
Public Website As String
```

Функции по работе с файлами

В VBA, помимо FSO, имеется ряд функций и операторов по работе с файлами и каталогами, дублирующие некоторые функции FSO. Одним из удобств этих функций является то, что для их применения не требуется устанавливать ссылку на используемую библиотеку. В табл. 13.13 эти функции и операторы перечислены.

Таблица 13.13. *Функции и операторы по работе с файлами и каталогами*

Функции и операторы	Описание
`ChDir`	Изменяет текущую папку. `ChDir` *`Path`*
`ChDrive`	Изменяет текущий диск. `ChDrive` *`Drive`*

Таблица 13.13 *(окончание)*

Функции и операторы	Описание
`CurDir`	Возвращает текущую папку
`Dir`	Возвращает файл или папку, соответствующую образцу, указанному в параметре *`PathName`*. Если несколько файлов соответствуют образцу, то возвращается первый из них. Используется для проверки существования файла на диске. `Dir[[(`*`PathName`*`[, `*`Attributes`*`])]]` Здесь параметр *`Attributes`* может принимать следующие значения: `vbNormal` или `0` (обычный файл), `vbHidden` или `2` (скрытый), `vbSystem` или `4` (системный), `vbVolume` или `8` (метка тома, если это значение указано, то все остальные атрибуты игнорируются), `vbDirectory` или `16` (каталог). В коде допустим вызов процедуры `Dir` без параметров для повторного поиска файла
`FileAttr`	Режим работы файла
`GetAttr`	Возвращает значение типа `Integer`, определяющее атрибуты файла или каталога
`SetAttr`	Устанавливает атрибуты файла
`FileCopy`	Копирует файл
`FileDateTime`	Возвращает дату и время последнего изменения файла
`Kill`	Удаляет существующий файл
`MkDir`	Создает новую папку
`RmDir`	Удаляет существующую папку

Поиск файла и объект *FileSearch*

Объект `FileSearch` предоставляет функциональные возможности окна **Открытие документа** по поиску файла. С данным объектом тесно связаны следующие семейства объектов:

- `FileTypes` — семейство всех типов файлов, среди которых производится поиск;
- `FoundFiles` — семейство всех найденных файлов;
- `PropertyTests` — семейство всех критериев поиска;
- `SearchScopes` — семейство подкаталогов, в которых производится поиск.

У объекта FileSearch имеется обширная коллекция свойств, перечисленных в табл. 13.15, способных задать различные параметры поиска, и всего три метода, приведенные в табл. 13.14, которые предназначены для инициализации и запуска поиска.

Таблица 13.14. *Методы объекта* `FileSearch`

Метод	Описание
`Execute`	Начать поиск указанного файла
`NewSearch`	Установить у всех параметров поиска значения, подразумеваемые по умолчанию
`RefreshScopes`	Обновить текущий список достижимых объектов `RefreshScope`

Таблица 13.15. *Свойства объекта* `FileSearch`

Свойство	Описание
`FileName`	Имя искомого файла, которое может включать символы "*" и "?"
`FileType`	Тип искомого файла
`FileTypes`	Возвращает список допустимых типов файла
`FoundFiles`	Возвращает объект `FoundFiles`, содержащий имена всех найденных файлов
`LastModified`	Возвращает константу, указывающую время последней модификации искомого файла. Допустимы значения: `msoLastModifiedAnyTime`, `msoLastModifiedLastMonth`, `msoLastModifiedLastWeek`, `msoLastModifiedThisMonth`, `msoLastModifiedThisWeek`, `msoLastModifiedToday`, `msoLastModifiedYesterday`
`LookIn`	Возвращает каталог, в котором производится поиск
`MatchAllWordForms`	Критерий сравнения слов при поиске файла, содержащем указанное слово
`MatchTextExactly`	Поиск файла, содержащего указанное слово
`PropertyTests`	Возвращает семейство `PropertyTests`, содержащее все критерии поиска файла
`SearchFolders`	Возвращает семейство `SearchFolders`, содержащее в себе все каталоги, в которых производится поиск
`SearchScopes`	Возвращает семейство `SearchScopes`

Таблица 13.15 *(окончание)*

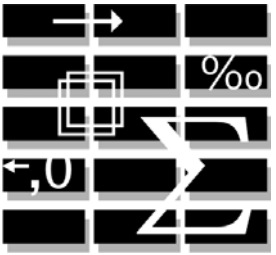
Свойство	Описание
SearchSubFolders	Устанавливает, надо ли производить поиск в подкаталогах
TextOrProperty	Задает слово или фразу, которая ищется в файле

Следующий код (листинг 13.27) демонстрирует, как можно произвести поиск файлов с указанным расширением в специфицированном каталоге и всех его подкаталогах.

Листинг 13.27. Поиск файлов

```
Sub DemoFileSearch()
   With Application.FileSearch
      .NewSearch
      .LookIn = "C:\jbproject"
      .SearchSubFolders = True
      .Filename = "*.jpx"
      .MatchTextExactly = True
      .FileType = msoFileTypeAllFiles
      If .Execute() > 0 Then
         Debug.Print "Найдено " & .FoundFiles.Count & " файлов"
         Dim i As Integer
         For i = 1 To .FoundFiles.Count
            Debug.Print .FoundFiles(i)
         Next
      Else
         Debug.Print "Файлы не найдены"
      End If
   End With
End Sub
```

ГЛАВА 14



Базы данных

База данных (database) — это набор данных, обычно представленных в виде записей. В качестве базы данных, вообще говоря, может выступать любой организованный набор данных, начиная от простого текстового файла и заканчивая сложной системой учета товаров, распределенной между несколькими компьютерами. Самым привычным примером базы является телефонная книга, содержащая имена, телефоны, коды и адреса. Каждая строка телефонной книги представляет собой запись, хранящую сведения об абоненте.

Согласно терминологии баз данных, *таблица* (table) — это двумерный массив данных. На рис. 14.1 показана простейшая база данных — телефонная книга, созданная в текстовом редакторе. Эта база данных состоит из единственной таблицы, в которой имеется пять столбцов: **Фамилия**, **Имя**, **Код**, **Телефон**, **Адрес**. Каждая из колонок описывает одно из информационных свойств объекта, хранимого в базе данных. Строка содержит фактические данные, т. е. она подробно описывает один хранимый объект и часто называется *записью* (record). Столбец же в таблице называется *полем* (field). Так,

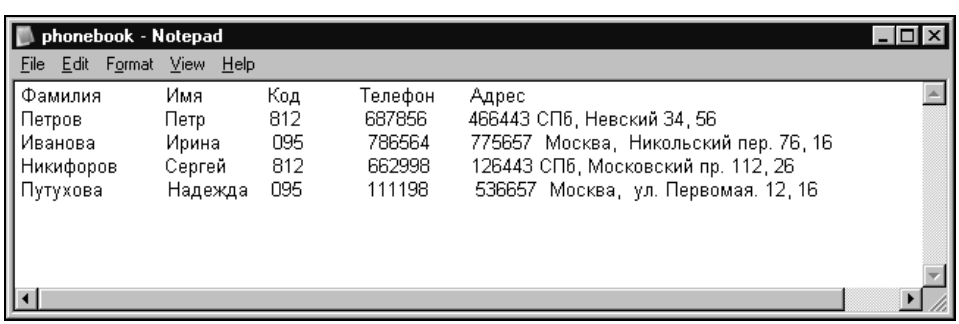


Рис. 14.1. Простейшая база данных — телефонная книга

таблица из рис. 14.1 имеет пять полей и пять записей. Каждая запись является уникальным элементом данных и может содержать столько полей, сколько требует бизнес-логика проекта. Например, в телефонную книгу можно добавить поля **Страна**, **Код страны**, **Электронный адрес**.

Приведенная база данных состоит всего из одной таблицы. В большинстве случаев базы данных состоят из большего числа таблиц, каждая из которых представляет объект какого-то типа. Наличие нескольких таблиц позволяет разработчику определить соотношение между информацией более эффективно и логично.

Предположим, что в телефонной книге требуется хранить следующую информацию об адресатах, которая организована в виде таблицы:

- ❒ Фамилия;
- ❒ Имя;
- ❒ Город;
- ❒ Почтовый индекс;
- ❒ Адрес;
- ❒ Код;
- ❒ Телефон.

Конечно, допустимо хранить всю эту информацию в одной таблице — так называемой *плоской базе данных* (flat-file database), но такая структура приводит к многократному повторению одной и той же информации. Например, Санкт-Петербург имеет один и тот же код, зачем же его повторять для каждого адресата, проживающего в Санкт-Петербурге. Многократно повторяющаяся информация требует много места для хранения базы данных и большего времени для обработки хранящейся в ней информации. Кроме того, плоские базы данных неизбежно ведут к ошибкам ввода, связанным с многократным вводом одних и тех же данных. Все мы просто люди, а не супермены, и потому рано или поздно при таком вводе получится опечатка. Более разумно хранить данные в двух таблицах.

Одна из них, которую назовем **Абонент**, содержит информацию о фамилии, имени, городе, почтовом индексе, адресе. Вторая же, которую назовем **Город**, содержит данные, относящиеся к городу, а именно название и код.

Таблица Абонент

```
Фамилия
Имя
Город
ПочтовыйИндекс
Адрес
Телефон
```

Таблица Город

Город
Код

Между этими двумя таблицами существует отношение по полю Город. Это поле присутствует в обеих таблицах. Отношение между таблицами **Город** и **Абонент** является отношением "один-ко-многим", потому что для каждой записи из таблицы **Город** существует либо одна, либо ни одной, либо много записей из таблицы **Абонент**. Такой тип базы данных, в которой таблицы связаны между собой, называется *реляционной базой данных* (relational database).

Обратите внимание на то, что в именах полей отсутствуют пробелы. Хотя механизм работы с базами данных позволяет в именах полей использовать пробелы и специальные символы, во избежание путаницы их предпочтительно не использовать, а имя образовывать по стандартному правилу имен переменных для VBA.

Ключи и индексы

Задача разработчика баз данных состоит в структурировании данных таким образом, чтобы устранить ненужное дублирование и обеспечить быстрый путь поиска всей необходимой информации. Для этих целей информацию делят на структурные единицы — таблицы, а сам процесс этого деления называется *нормализацией*. Ключевым правилом нормализации является следующее: таблицы, которые содержат повторяющуюся информацию, должны быть разделены на отдельные таблицы, чтобы устранить дублирование значений. Например, если бы база данных состояла из одной таблицы, содержащей данные как по заказам, так и по заказчикам, то данные заказчиков во многих записях повторялись бы. После разделения этой таблицы на две таблицы дублирование данных устраняется при существенном сокращении базы данных. Кроме того, устранение повторяемости данных, как уже отмечалось ранее, уменьшает вероятность ввода ошибочных данных.

Итак, при составлении таблиц важно исключить дублирование записей. Здесь на помощь приходят так называемые *первичные ключи* (primary keys). Первичный ключ должен уникальным образом идентифицировать запись. Первичный ключ может содержать значащие данные или же может быть произвольным полем, созданным только для того, чтобы снабдить запись неповторяющимся идентификатором. Главное требование для первичного ключа — уникальность. Например, если в таблице с телефонной книгой **Абоненты** объявить поле Фамилия в качестве первичного ключа, то после этого таблица не сможет содержать две записи с одинаковыми фамилиями. Вообще говоря, в качестве первичного ключа можно объявить несколько полей, например Фамилия и Имя. Тогда в таблице не смогут содержаться две

записи о людях с одной и той же фамилией и именем одновременно. Это более совершенный подход к созданию первичного ключа, но он все еще не безопасен, особенно для больших телефонных книг. В большом городе с высокой вероятностью существуют однофамильцы с одним и тем же именем. Лучшим способом обеспечения отсутствия дублирующих записей был бы ввод дополнительного *идентификационного поля* (identity field). В качестве наиболее простого способа идентификации по этому полю может быть номер записи в телефонной книге. Итак, в таблице **Абонент** должно появиться новое поле `ID`, однозначно идентифицирующее запись.

Таблица Абонент

```
ID
Фамилия
Имя
Город
ПочтовыйИндекс
Адрес
Телефон
```

Только после создания такого поля можно переходить к созданию связи с другой таблицей, в данном случае с таблицей **Город**. Это делается путем установки отношения между полями в различных таблицах. Каждое такое поле должно содержать одинаковую информацию. Таблицы **Абонент** и **Город** связаны по полю `Город`. Поле в таблице, которое хранит значение, связанное со значением другой таблицы, называется *внешним ключом* (foreign key).

Для быстроты обработки данных в таблицах используют *индексы* (index). Индекс — это атрибут, который можно присвоить полю, чтобы облегчить для процессора базы данных выборку данных на основе информации, хранимой в этом поле. Например, в телефонной книге, вероятно, будет реализована функция поиска записи по фамилии или имени. Поэтому для каждого из этих полей имеет смысл создать индексы, чтобы ускорить процесс выборки записей на их основе. Но не следует чрезмерно перегружать таблицы индексами. При увеличении количества индексов увеличивается размер базы данных, что может свести на нет положительный эффект от использования индексов.

База данных *Борей*

Хорошим примером базы данных с несколькими таблицами может служить учебная база данных **Борей**, хранимая в файле Борей.mdb и поставляемая вместе с MS Office. Давайте откроем эту базу в MS Access для того, чтобы взглянуть на ее структуру (рис. 14.2).

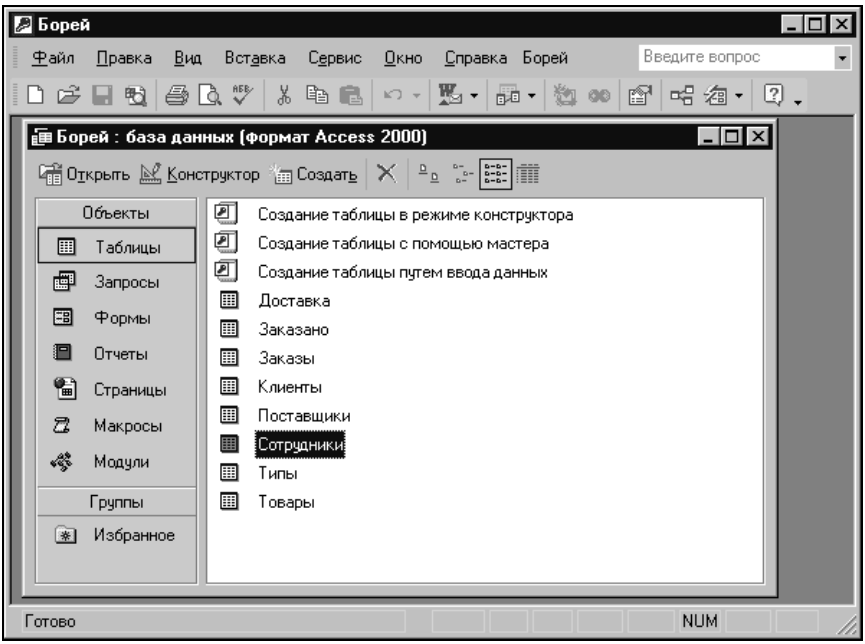


Рис. 14.2. Учебная база данных **Борей**

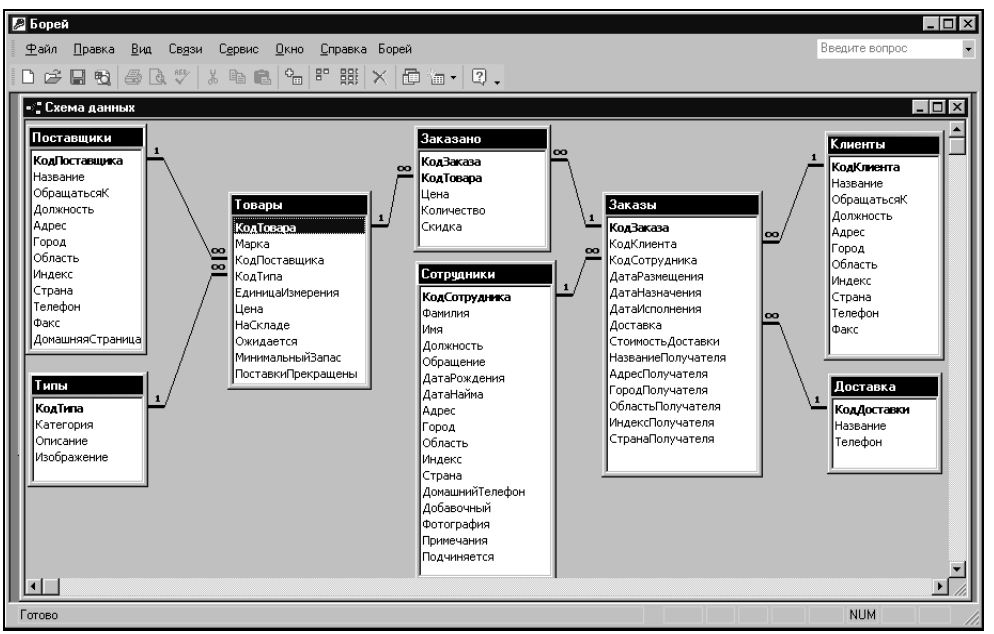


Рис. 14.3. Схема данных

В базе данных **Борей** имеется восемь таблиц. Уже из их названий ясно, что некоторые из этих таблиц, скорее всего, связаны между собой, например **Заказано** и **Заказы**. Чтобы получить более полную информацию о связях, выберите команду **Сервис | Схема данных**. На экране отобразится окно **Схема данных** (рис. 14.3).

Отношения

Для связывания таблиц между собой используются *отношения*. Отношения гарантируют сохранение *целостности данных* (data integrity). Целостность данных гарантирует, что пользователь будет работать только с непротиворечивыми данными и разрешенными в отношении связанных таблиц операциями. К примеру, при соблюдении целостности данных вы сможете добавить записи заказов только для существующих клиентов.

Между таблицами в базе данных можно установить отношения трех типов:

- "Один-ко-многим". Это отношение обозначается в окне **Схема данных** линией 1 — ∞. Например, таблицы **Клиенты** и **Заказы** связаны отношением типа "один-ко-многим". Это означает, что один клиент может сделать несколько заказов.
- "Один-к-одному". Это отношение обозначается в окне **Схема данных** линией 1 — 1. Отношение типа "один-к-одному" возникает тогда, когда с каждой стороны отношения имеется ровно одна запись, например когда информация о финансовом состоянии клиента должна быть отдельно от остальной информации о нем.
- "Многие-ко-многим". Это отношение обозначается в окне **Схема данных** линией ∞ — ∞ и возникает тогда, когда каждая запись первой таблицы связана со многими записями второй таблицы и наоборот. В подобных случаях обычно строится промежуточная таблица, обеспечивающая отношение "один-ко-многим" с каждой из таблиц по отдельности. Классическим примером такого отношения является таблица товаров, приобретенных покупателями. Каждый покупатель может приобрести множество различных товаров, и каждый товар может быть куплен разными покупателями.

Создание базы данных в MS Access

MS Access обладает высокоорганизованным и простым интерфейсом для создания объектов баз данных, он позволяет управлять всеми данными из одного файла базы данных. В рамках этого файла используются следующие объекты:

- таблицы для сохранения данных;
- запросы для поиска и извлечения только требуемых данных;

- формы для просмотра, добавления и изменения данных в таблицах;
- отчеты для анализа и печати данных в определенном формате;
- страницы доступа к данным для просмотра, обновления и анализа данных из базы через Internet или интрасеть.

В этом и следующем разделах демонстрируется процесс создания базы данных в MS Access на примере телефонной книги, состоящей из двух таблиц. Итак, приступим. Прежде всего надо создать файл базы данных. Для этого:

1. Запустите MS Access 2002.
2. Выберите команду **Файл | Создать**.
3. В появившейся панели инструментов **Создание файла** нажмите кнопку **Новая база данных**.
4. На экране отобразится окно **Файл новой базы данных**. В поле **Имя файла** введите `phonebook`, а в списке **Папка** выберите требуемое местоположение файла. Нажмите кнопку **Создать**. Файл базы данных создан.

Создание таблицы базы данных в MS Access

Итак, файл базы данных создан. Перейдем к конструированию сначала таблицы **Абонент**, а затем таблицы **Город**. После открытия файла базы данных интегрированная среда разработки предлагает три способа конструирования таблицы (рис. 14.4):

- создание таблицы в режиме конструктора;
- создание таблицы с помощью мастера;
- создание таблицы в процессе ввода данных.

Рассмотрим создание таблицы в режиме конструктора.

1. Выберите способ **Создание таблицы в режиме конструктора**. Появится окно конструктора таблиц (рис. 14.5). Перейдем к построению таблицы **Абонент**.
2. В первую строку столбца **Имя поля** введите имя первого поля `ID`.
3. Нажмите клавишу <Tab> для перехода к следующему столбцу.
4. Выберите тип данных **Счетчик**. При этом создается поле для хранения целого числа, которое при создании новой записи автоматически заполняется уникальным номером. Кроме типа **Счетчик** у полей в MS Access имеются типы данных, перечисленные в табл. 14.1.
5. Чтобы сделать это поле первичным ключом, выберите команду **Правка | Ключевое поле** или нажмите кнопку **Ключевое поле** панели инструментов **Конструктор таблиц** (рис. 14.6).
6. Создайте остальные поля таблицы (табл. 14.2 и рис. 14.7).

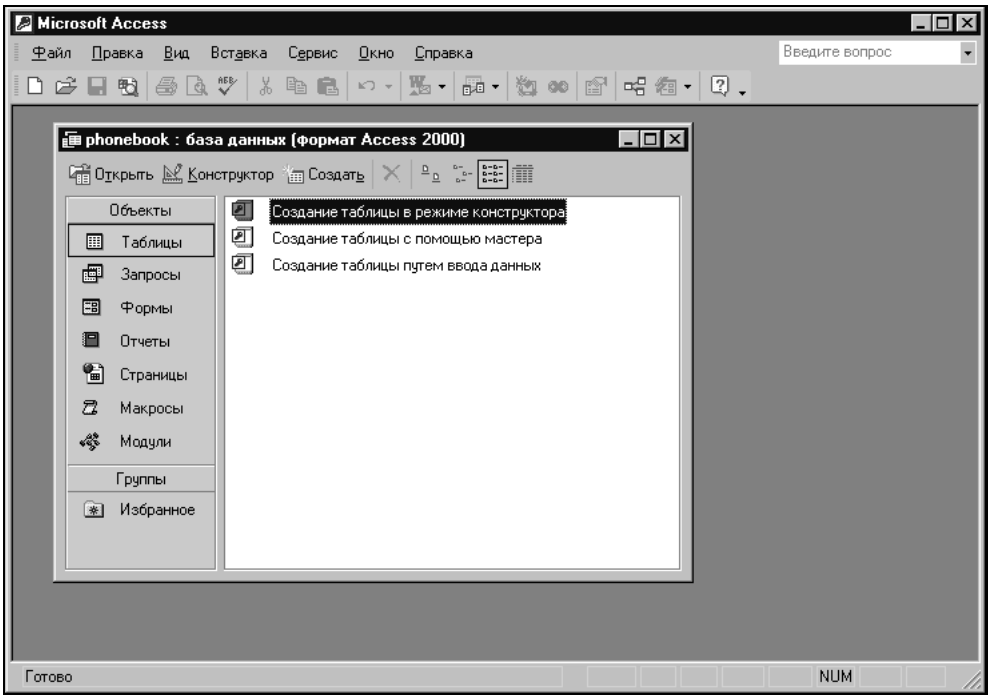


Рис. 14.4. Способы создания таблицы

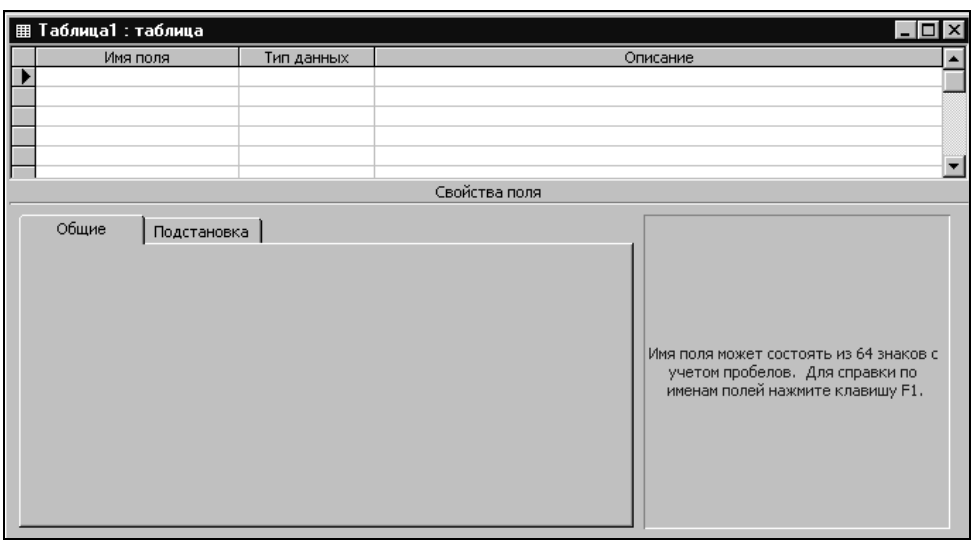


Рис. 14.5. Окно конструктора таблиц

Таблица 14.1. *Типы данных*

Тип данных	Описание
Текстовый	Текст или комбинация текста и чисел, например адреса, а также числа, не требующие вычислений, например номера телефонов, инвентарные номера или почтовые индексы. Сохраняет до 255 знаков. Свойство **Размер поля** определяет максимальное количество знаков, которые можно ввести в поле
Поле MEMO	Длинный текст или числа, например, примечания или описания. Сохраняет до 65 536 знаков
Числовой	Данные, используемые для математических вычислений, за исключением финансовых расчетов (для них следует использовать тип `Денежный`). Сохраняет 1, 2, 4 или 8 байтов; 16 байтов для кодов репликации (GUID). Конкретный тип числового поля определяется значением свойства **Размер поля**
Дата/Время	Значения дат и времени. Сохраняет 8 байтов. Допустимо хранение дат, лежащих в диапазоне между 100 и 9999 годами
Денежный	Используется для денежных значений и для предотвращения округления во время вычислений. Сохраняет 8 байтов
Счетчик	Автоматическая вставка уникальных последовательных (увеличивающихся на 1) или случайных чисел при добавлении записи. Сохраняет 4 байта; 16 байтов для кодов репликации (GUID)
Логический	Данные, принимающие только одно из двух возможных значений, таких как `Да/Нет`, `Истина/Ложь`, `Вкл/Выкл`. Сохраняет 1 бит
Поле объекта OLE	Объекты OLE (такие как документы MS Word, электронные таблицы MS Excel, рисунки, звукозаписи или другие данные в двоичном формате), созданные в других программах, использующих протокол OLE. Сохраняет до 1 Гбайт (ограничивается объемом диска)
Гиперссылка	Гиперссылка. Гиперссылка может иметь вид пути UNC либо адреса URL. Сохраняет до 64 000 знаков
Мастер подстановок	Создает поле, позволяющее выбрать значение из другой таблицы или из списка значений, используя поле со списком. При выборе данного параметра в списке типов данных запускается мастер для автоматического определения этого поля

Таблица 14.2. *Поля таблицы* ***Абонент*** *без поля* `ID`

Поле	Тип данных
`Фамилия`	`Текстовый`
`Имя`	`Текстовый`

Таблица 14.2 *(окончание)*

Поле	Тип данных
Город	Текстовый
ПочтовыйИндекс	Текстовый
Адрес	Текстовый
Телефон	Текстовый

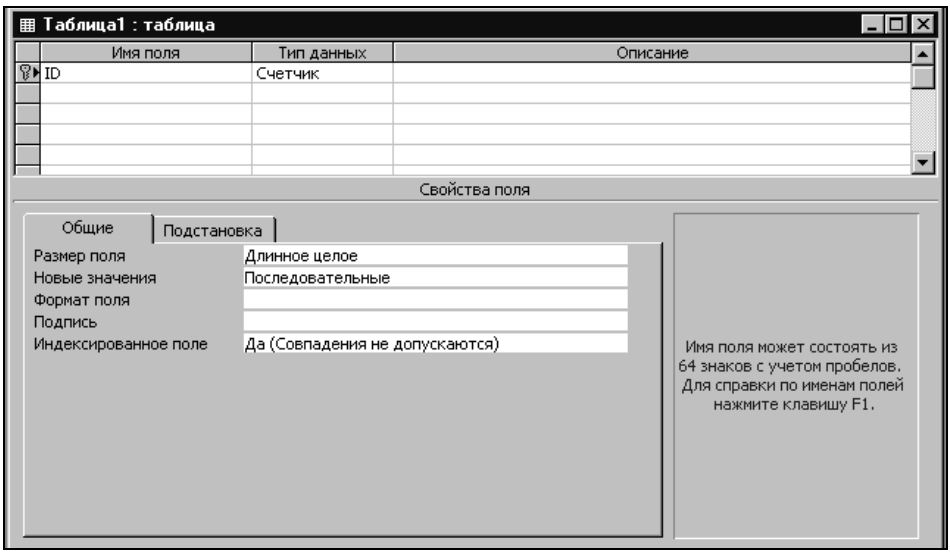


Рис. 14.6. Установленный первичный ключ

7. Закройте таблицу. Вам будет предложено ответить на вопрос о сохранении этой таблицы. Сохраните ее под именем **Абонент**.

Аналогично создайте таблицу **Город** с двумя полями, как показано в табл. 14.3.

Таблица 14.3. *Поля таблицы* ***Город***

Поле	Тип данных	Первичный ключ
Город	Текстовый	Да
Код	Текстовый	Нет

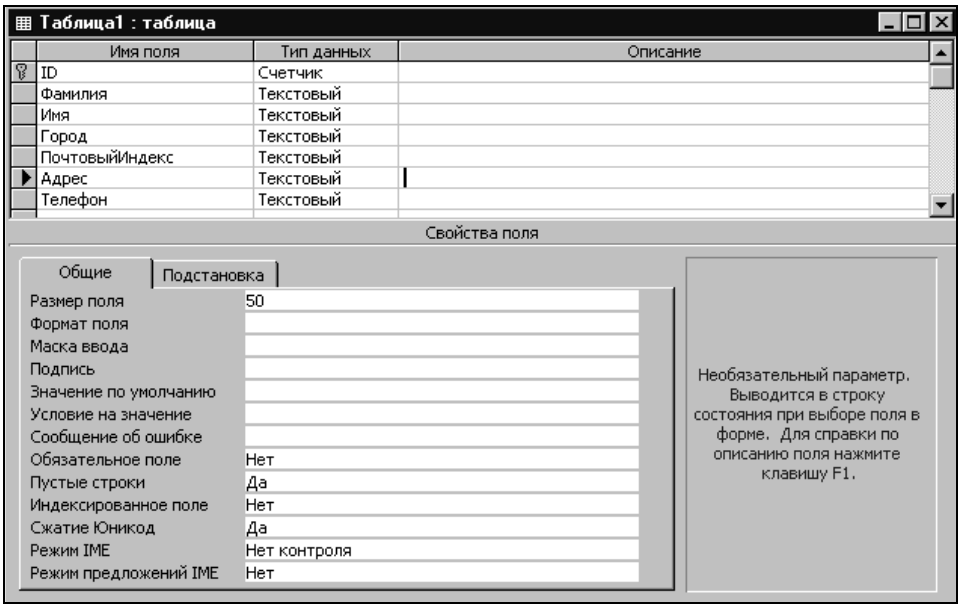


Рис. 14.7. Заполненная таблица

Создание отношений с помощью MS Access

После того как у нас появилась база данных с двумя связанными таблицами, можно продемонстрировать процесс установления соотношения между таблицами **Абонент** и **Город**.

1. В MS Access выберите команду **Сервис | Схема данных**.
2. Появится окно **Добавление таблицы** (рис. 14.8). При помощи кнопки **Добавить** добавьте в схему данных таблицы **Абонент** и **Город**, после чего нажмите кнопку **Закрыть**.
3. В окне **Схема данных** нажмите левую кнопку мыши на поле `Город` таблицы **Абонент** и, не отпуская мыши, перетащите его в поле `Город` таблицы **Город**. На экране отобразится окно **Изменение связей** (рис. 14.9).
4. Установите флажок **Обеспечение целостности данных**, после чего установите флажки **каскадное обновление связанных полей** и **каскадное удаление связанных записей**.
5. Нажмите кнопку **Создать**. MS Access создаст отношения, как показано на рис. 14.10.

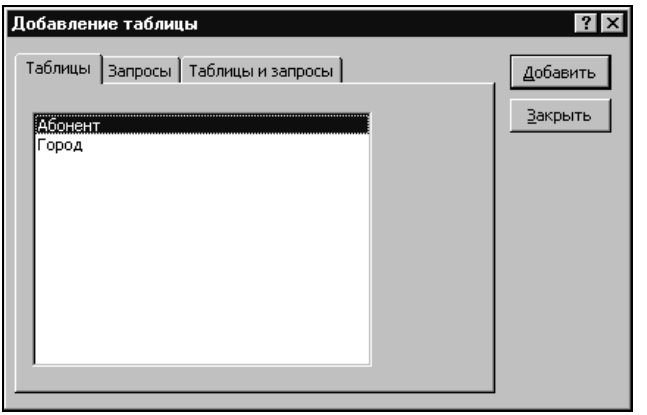


Рис. 14.8. Окно **Добавление таблицы**

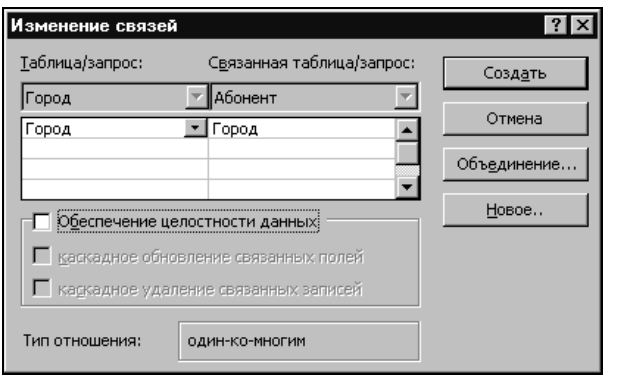


Рис. 14.9. Окно **Изменение связей**

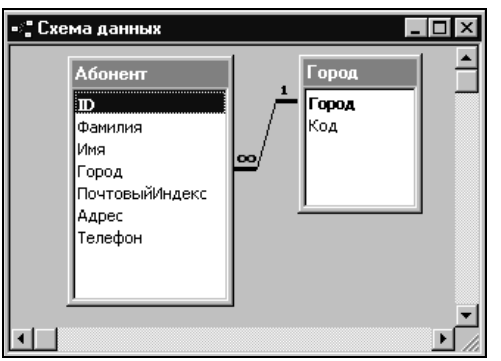


Рис. 14.10. Окно **Схема данных**

Запрос и язык запросов SQL

Запрос (query) — это команда базы данных, осуществляющая выборку записей. Используя запрос, можно получить данные из одного или нескольких полей, принадлежащих одной или нескольким таблицам. При этом можно также подвергать отбираемые данные действию одного или нескольких условий, называемых *критериями*, которые служат для ограничения общего объема отбираемых данных. Запросы в VBA обычно основаны на структурированном языке запросов SQL. Это стандартный язык для выборки информации и осуществления других операций над базами данных. Команды языка SQL делятся на две большие категории: команды манипулирования данными (DML) и команды определения данных (DDL). SQL-команды позволяют отображать для пользователя не всю таблицу, а какую-то выборку данных из этой таблицы или даже из нескольких таблиц. Причем, выборка может производиться по довольно сложным критериям, с одновременным подведением итогов по выбираемым данным. В последующих разделах показано, как при помощи команды манипулирования данными строятся итоговые таблицы.

Оператор *SELECT*

Итак, перейдем к обсуждению команд и операторов языка SQL. Оператор `SELECT` возвращает построенную выборку данных из одной или нескольких таблиц базы данных по указанным критериям.

Синтаксис оператора `SELECT` выглядит довольно устрашающе. Но пускай это вас не беспокоит. Сразу же после приведения его формального синтаксиса на конкретных примерах мы обсудим, как работает этот оператор.

```
SELECT [предикат] { * | таблица.* | [таблица.]поле_1
[AS псевдоним_2] [, [таблица.]поле_2 [AS псевдоним_2] [, ...]]}
FROM выражение [, ...]
[WHERE where-условие]
[GROUP BY groupby-поле]
[HAVING having-условие]
[ORDER BY orderby-поле[,orderby-поле]. ]
```

- *предикат* — предикаты используются для ограничения числа возвращаемых записей. Если они отсутствуют, по умолчанию используется предикат `ALL`. Допустимые значения:
  - `ALL` — возвращает все записи;
  - `DISTINCT` — исключает записи, которые содержат повторяющиеся значения в выбранных полях;

- DISTINCTROW — опускает данные, основанные на целиком повторяющихся записях, а не на отдельных повторяющихся полях;
- TOP — возвращает определенное число записей, находящихся в начале или в конце диапазона, описанного с помощью предложения ORDER BY;

- * — указывает, что выбраны все поля заданной таблицы;
- *таблица* — имя таблицы, из которой должны быть отобраны записи;
- *поле_n* — имена полей, из которых должны быть отобраны данные. Если включить несколько полей, они будут извлекаться в указанном порядке;
- *псевдоним_n* — имена, которые станут заголовками столбцов вместо исходных названий столбцов в таблице;
- *выражение* — имена одной или нескольких таблиц, которые содержат отбираемые данные;
- *where-условие* — условие выбора;
- *groupby-поле* — поле, по которому производится группировка;
- *having-условие* — условие подведения промежуточных итогов;
- *orderby-поле* — поле, по которому производится сортировка.

Итак, перейдем к последовательной иллюстрации действия оператора SELECT на примере следующей простой базы данных, состоящей из единственной таблицы **Абонентн** (табл. 14.4).

Таблица 14.4. *Таблица* ***Абонентн***

ID	Фамилия	Имя	Телефон	Город	Возраст
1	Иванов	Иван	123-32-34	Москва	36
2	Петров	Петр	342-43-34	Санкт-Петербург	24
3	Сидоров	Сидор	324-45-34	Москва	34
4	Иванов	Федор	355-45-34	Киев	45
5	Кривошеева	Татьяна	454-45-45	Самара	45
6	Курносова	Ирина	345-45-43	Санкт-Петербург	56
7	Сидкин	Петр	123-34-45	Санкт-Петербург	12
8	Иванов	Игорь	123-34-45	Санкт-Петербург	34

Возвращение всех записей таблицы

Если требуется вернуть все записи таблицы (в данном случае **Абонентн**), то можно воспользоваться следующей командой:

```
SELECT * FROM Абонентн
```

или, что равносильно,

```
SELECT ALL * FROM Абонентн
```

Возвращение всех записей одного поля

Для того чтобы вернуть набор всех записей одного поля, например Фамилия, примените следующую команду:

```
SELECT Фамилия FROM Абонентн
```

Возвращение всех записей двух полей таблицы

Если требуется вернуть все записи двух полей таблицы, например Фамилия и Телефон, то можно воспользоваться следующей командой:

```
SELECT Фамилия, Телефон FROM Абонентн
```

Упорядочивание элементов и предложение *ORDER BY*

Добавление в SQL-команду предложения ORDER BY обеспечивает упорядочивание набора выбранных записей. Добавление ключевого слова ASC или DESC после имени поля задает порядок сортировки по этому полю (возрастающий или убывающий).

Упорядочивание записей по одному полю

Следующий оператор возвращает всю таблицу **Абонентн**, записи которой упорядочены по полю Фамилия в порядке возрастания:

```
SELECT * FROM Абонентн ORDER BY Фамилия ASC
```

Упорядочивание записей по двум полям

Приводимый ниже оператор возвращает записи двух полей (Фамилия и Рост) таблицы **Абонентн**, причем записи упорядочены по полю Фамилия в порядке возрастания, а по полю Возраст в порядке убывания:

```
SELECT Фамилия, Рост FROM Абонентн ORDER BY Фамилия ASC, Возраст DESC
```

Выборка записей по критерию и предложение *WHERE*

Добавление в SQL-команду предложения WHERE обеспечивает проведение выборки по критерию. В предложении WHERE допустимо использование простых операций сравнения (=, <>, >, <, >=, <=), например

Фамилия = 'Иванов' или Возраст > 20, либо составных с использованием логических операторов AND и OR.

Ссылка на строки и даты в операциях сравнения

Обратите внимание на то, что все строки в операциях сравнения заключаются в простые кавычки, например, Фамилия = 'Иванов'. Подобным же образом даты должны обрамляться символом "#", например, 'ДатаРождения' > #12/7/99#.

Предложение *WHERE* и выборка записей по простому критерию

Добавление в SQL-команду предложения WHERE обеспечивает проведение выборки по критерию. В предложении WHERE допустимо использование простых операций сравнения (=, <>, >, <, >=, <=). Например, первая из следующих двух команд возвращает все записи с московскими телефонами, а вторая — всех тех адресатов, которые не младше 20.

```
SELECT * FROM Абоненти WHERE Город = 'Москва'
SELECT * FROM Абоненти WHERE Возраст >= 20
```

Выборка записей по составному критерию

В предложении WHERE допустимо применение составных операций с использованием логических операторов AND и OR. Например, следующая команда возвращает как все записи с московскими телефонами, так и те, у которых возраст абонентов лежит в диапазоне от 21 до 30.

```
SELECT * FROM Абоненти
↳WHERE (Город = 'Москва') OR (Возраст >= 21 And Возраст <= 30)
```

Выборка записей, значения специфицированного поля которых принадлежат указанному диапазону или лежат за его пределами

В конструкции WHERE допустимо использование операций сравнения BETWEEN AND или NOT BETWEEN AND — для выборки записей, значения специфицированного поля которых принадлежат указанному диапазону или лежат за его пределами. В первом из следующих двух примеров возвращаются все записи с возрастом абонентов от 21 до 30, а во втором — вне этого диапазона.

```
SELECT * FROM Абоненти WHERE (Возраст BETWEEN 21 And 30)
SELECT * FROM Абоненти WHERE (Возраст NOT BETWEEN 21 And 30)
```

Выборка записей, значения специфицированного поля которых принадлежат указанному множеству

В конструкции WHERE допустимо использование операций сравнения IN для выборки записей, значения специфицированного поля которых принадлежат указанному множеству. В следующем примере возвращаются все записи с московскими и петербургскими телефонами.

```
SELECT * FROM Абонентн WHERE (Город In ('Москва','Санкт-Петербург'))
```

Выборка записей по шаблону

В конструкции WHERE допустимо использование операций сравнения LIKE для выборки записей, значения специфицированного поля которых имеют определенный шаблон. При создании шаблонов можно использовать символ "%" для указания множества символов. Операция NOT LIKE задает все записи, не подходящие к указанному шаблону. Например, первая из следующих двух команд возвращает все записи с фамилиями, оканчивающимися на "ва", т. е. записи с женскими фамилиями, а вторая команда — все остальные.

```
SELECT * FROM WHERE Фамилия Like '%ва'
SELECT * FROM Абонентн WHERE Фамилия NOT LIKE %ва'
```

Выбор различных записей

Использование предикта DISTINCT в операторе SELECT позволяет отображать только неодинаковые записи. Например, следующая команда выводит всех абонентов с различными фамилиями.

```
SELECT DISTINCT Фамилия FROM Абонентн
```

Нахождение общего числа записей, возвращаемого запросом

Для определения количества записей, удовлетворяющих специфицированному критерию, достаточно воспользоваться функцией COUNT со значением * в качестве ее параметра. Например, следующая команда возвращает число московских абонентов из телефонной книги.

```
"SELECT COUNT(*) FROM Абонентн WHERE Город='Москва'
```

Нахождение общего числа непустых записей

Для определения количество непустых записей надо воспользоваться функцией COUNT, в качестве параметра которой указывается поле, непустые элементы которого считаются, например, как это делается в следующей

команде, где находится количество всех абонентов с указанными фамилиями, возраст которых больше 25.

```
SELECT COUNT(Фамилия) FROM Абонентн WHERE Возраст > 25
```

Статистические функции

Оператор SELECT можно применять для выполнения вычислений со значениями, которые хранятся в таблицах. При этом используются статистические функции, приведенные в табл. 14.5, которые вычисляют соответствующие значения поля в записях, отвечающих условию WHERE.

Таблица 14.5. *Итоговые функции языка SQL*

Функция	Возвращаемое значение
SUM([DISTINCT] *exprs*)	Сумма (неповторяющихся) значений
AVG([DISTINCT] *exprs*)	Среднее значение (неповторяющихся) величин
COUNT([DISTINCT] *exprs*)	Число (неповторяющихся) значений, не равных NULL
COUNT(*)	Число выбранных строк
MIN(*exprs*)	Минимальное значение в данном выражении
MAX(*exprs*)	Максимальное значение в данном выражении

Например, следующая команда возвращает число московских абонентов, а также их минимальный, максимальный и средний возраст.

```
SELECT COUNT(Возраст), MIN(Возраст), MAX(Возраст), AVG(Возраст)
↳FROM Абонентн WHERE Город='Москва'
```

Определение, пусто ли поле, и значение *NULL*

Значение NULL, используемое в SQL, не только отражает отсутствие информации в ячейке таблицы, но и указывает на то, что некоторая информация неизвестна. В каком-то смысле это значение переводит SQL в язык трехзначной логики, т. к. возвращаемым значением может быть как "истина" или "ложь", так и неопределенная величина, т. е. NULL. Определить, содержит ли поле NULL, можно при помощи операции сравнения IS, а то, что там такового нет — при помощи операции сравнения IS NOT. Например, следующая команда создает набор всех записей с непустым полем Возраст.

```
SELECT Фамилия, Возраст FROM Абонентн WHERE Возраст IS NOT NULL
```

Оператор *DELETE*

Оператор DELETE применяется для создания функционального запроса на удаление из таблицы определенных записей. Функциональный запрос не возвращает группу записей в виде динамического набора, как это делает оператор SELECT. Он работает аналогично программе.

Оператор DELETE имеет следующий синтаксис:

```
DELETE FROM имяТаблицы [WHERE условие]
```

Действие оператора DELETE нельзя отменить. Например, следующий код удаляет из таблицы **Абонентн** запись, у которой в поле Фамилия введена Иванова.

```
DELETE FROM Абонентн WHERE Фамилия='Иванова'
```

Оператор *INSERT*

Подобно DELETE, оператор INSERT создает функциональный запрос. В сочетании с оператором SELECT он используется для добавления в таблицу новых записей. Оператор INSERT имеет следующий синтаксис:

```
INSERT INTO имяТаблицы [(списокСтолбцов)] VALUES(списокДанных)
```

Например, следующая команда вставляет в поля Фамилия и Телефон таблицы **Абонентн** значения Иванова и 111-11-11.

```
INSERT INTO Абонентн(Фамилия, Телефон) VALUES ('Иванова','111-11-11')
```

Оператор *UPDATE*

Еще одной разновидностью функционального запроса является оператор UPDATE. С его помощью изменяются значения указанных полей таблицы. Оператор UPDATE имеет следующий синтаксис:

```
UPDATE имяТаблицы SET [поле=значение[,поле=значение]...] [WHERE условие]
```

Например, следующая инструкция обновляет запись со значением поля **Фамилия**, равным Иванова.

```
UPDATE Абонентн SET Телефон='121-11-11', Возраст = 31
↳WHERE Фамилия='Иванова'
```

Создание групп записей

Возможность создания групп записей позволяет построить набор записей, в котором для каждого значения указанного поля будет только одна запись. Например, если группировать таблицу **Абонентн** по городам, то для каждой

страны получится одна выходная запись. Такая организация записей особенно удобна в сочетании с описанными выше статистическими функциями. В этом случае легко получить итоговую информацию для городов, определенного товара, торгового агента и т. д. Например, следующий запрос возвращает суммарный возраст абонентов в каждом городе.

```
SELECT Город, SUM(Возраст) AS [Sum-Возраст] FROM Абонентн GROUP BY Город
```

Создание псевдонима с использованием предложения *AS*

В SQL существует возможность переименования полей в запросе посредством определения их псевдонимов (alias) при помощи предложения `AS`. Псевдонимы используются по двум причинам:

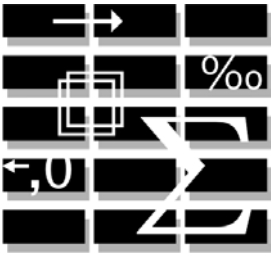
- если необходимо упростить громоздкие имена полей;
- если запрос создает столбец (как это и произошло в примере из предыдущего раздела), который заполняется в результате некоторых вычислений, и новому столбцу необходимо задать имя.

Предложение *HAVING*

Предложение `HAVING` позволяет создать результирующие наборы, содержащие список промежуточных итогов. Например, следующий запрос возвращает суммарный возраст абонентов в каждом городе при условии наличия более одного абонента.

```
SELECT Город, SUM(Возраст) AS [Sum-Возраст] FROM Абонентн GROUP BY Город
↳HAVING COUNT(Возраст) > 1
```

ГЛАВА 15



ADO (ActiveX Data Objects)

Технология ADO (ActiveX Data Objects) предоставляет разработчику универсальный метод извлечения и изменения данных из разнообразных источников, которые не ограничиваются только традиционными базами данных. Ими могут быть и файловая система, и Internet. Рука об руку с ADO идет OLE DB — набор интерфейсов COM (Component Object Model, компонентная модель объектов), используемых для извлечения данных. OLE DB предоставляет низкоуровневое подключение к данным через интерфейс COM, в то время как ADO предоставляет модель объектов, которая упрощает процесс использования OLE DB и написания кода. О работе с базами данных в MS Excel при помощи технологии ODBC рассказано в моей книге "Excel, VBA, Internet в экономике и финансах", опубликованной издательством "БХВ-Петербург".

Технология ADO позволяет решать следующие задачи:

- создавать соединение с источником данных;
- создавать объект, реализующий SQL-команду;
- выполнять SQL-команду;
- сохранять результат выполнения SQL-команды в кэше;
- сконструировать виртуальное представление кэша, чтобы пользователь мог фильтровать, сортировать данные и перемещаться по ним;
- редактировать данные;
- обновлять источник данных в соответствии с внесенными изменениями;
- фиксировать или отменять изменения, внесенные в процессе транзакции и последующего ее закрытия.

В объектную модель ADO входят перечисленные в табл. 15.1 объекты.

Таблица 15.1. *Объектная модель ADO*

Объект	Описание
`Connection`	Среда, в которой выполняется обмен данными с источником данных. Соединение с источником данных необходимо создать до начала выполнения любых операций
`Command`	Способ управления источником данных. Позволяет добавлять, удалять и считывать данные из источника
`Parameter`	Переменные компоненты объекта `Command`, которые уточняют способ выполнения команд. Перед выполнением каждой из команд эти параметры можно модифицировать
`Recordset`	Локальный кэш для данных, считанных из источника
`Field`	Столбец таблицы данных `Recordset`. Поля содержат свойства, определяющие их параметры, например тип
`Error`	Инкапсулирует данные о сгенерированной ошибке
`Property`	Определяет объекты `Connection`, `Command`, `Recordset` и `Field`. Каждый объект ADO обладает набором свойств, описывающих его или им управляющих. Встроенные свойства существуют всегда. Динамические свойства добавляются источником данных OLE DB и существуют только до конца работы с ним
`Collection`	Служит для объединения сходных объектов ADO в группы. Имеются четыре подобных группы объектов: `Errors`, `Parameters`, `Fields` и `Properties`. Каждый объект `Recordset` включает в себя семейство `Fields`, представляющее все поля таблицы. Объект же `Command` содержит семейство `Parameters`, возвращающее параметры команды
`Stream`	Поток ввода/вывода текста или данных

Создание ссылки на библиотеку ADO

Прежде чем применять технологию ADO на компьютере, должна быть инсталлирована библиотека ADO. Она инсталлируется вместе с Microsoft Office и может функционировать в любой среде разработки, в которой поддерживается взаимодействие с объектами ActiveX/COM, включая Visual Basic, Visual C++ и Active Server Pages.

Если библиотека ADO инсталлирована, то ее объекты можно использовать после установки ссылки на эту библиотеку. Для этого:

1. Выберите команду **Tools | References**. Появится окно **References**.
2. Установите флажок **Microsoft ActiveX Data Objects 2.0 Library**. Нажмите кнопку **OK**.

После выполнения этих действий ADO может быть использована в вашем коде.

Объект *Connection* и установка подключения к базе данных

Объект `Connection` представляет собой среду, в которой выполняется обмен данными с источником данных. Соединение с источником данных необходимо создать до начала выполнения любых операций. После прекращения обмена данными с источником, соединение должно быть закрыто.

Для установки подключения к источнику данных применяется метод `Open` объекта `Connection`, а для закрытия соединения — метод `Close`.

Свойство `Provider` является текстовой строкой, задающей тип провайдера OLE DB, который будет применен для подключения. При использовании провайдера ODBC его можно не указывать, т. к. он используется по умолчанию и называется `MSDASQL`.

Свойство `ConnectionString` указывает способ подключения к источнику данных.

При использовании Jet-провайдера строка подключения может состоять из полного имени файла. Например, в следующем коде (листинг 15.1) при инициализации формы устанавливается соединение с базой данных **Борей** с использованием Jet-провайдера, а при закрытии формы — это соединение также закрывается.

Листинг 15.1. Подключение с использованием Jet-провайдера

```
Private cn As ADODB.Connection

Private Sub UserForm_Initialize()
   Set cn = New ADODB.Connection
   cn.Provider = "Microsoft.Jet.OLEDB.4.0"
   cn.ConnectionString = "c:\Борей.mdb"
   cn.Open
End Sub

Private Sub UserForm_Terminate()
   cn.Close
End Sub
```

При использовании провайдера ODBC строка подключения может задавать имя источника данных DSN (Data Source Name). Например, в следующем коде (листинг 15.2) при инициализации формы устанавливается соединение с базой данных **Борей**, у которой DSN установлен равным `br`.

Листинг 15.2. Подключение с использованием ODBC-провайдера

```
Private Sub UserForm_Initialize()
   Set cn = New ADODB.Connection
   cn.Provider = "MSDASQL"
   cn.ConnectionString = "DSN=br"
   cn.Open
End Sub
```

Установка DSN

Задать имя источника данных DSN можно следующим образом:

1. Выберите команду **Пуск | Настройки | Панель управления**.
2. Дважды щелкните на значке **Источник данных ODBC (32-разрядный)**, расположенном в открытой папке **Панель управления**.
3. Появится окно **Администратор источников данных ODBC**. Выберите вкладку **Системный DSN**. В приводимом ранее примере использовалась учебная база **Борей**, поэтому сейчас создадим для нее источник данных. Нажмите кнопку **Добавить**. Появится окно **Создание нового источника данных**. Выберите драйвер, для которого создается источник, в данном случае **Microsoft Access Driver (*.mdb)**. Нажмите кнопку **Готово**.
4. Появится окно **Установка драйвера ODBC для Microsoft Access**. В поле **Имя источника данных** введите, например, `br`. В поле **Описание** введите описание этого источника. Нажмите кнопку **Выбрать**. На экране появится окно **Выбор базы данных**. Выберите файл Борей.mdb из папки, где установлен MS Office. Нажмите кнопку **OK**. Окно **Выбор базы данных** закроется. Нажмите кнопку **OK**. Окно **Установка драйвера ODBC для Microsoft Access** закроется. Теперь на вкладке **Системный DSN** окна **Администратор источников данных ODBC** появился новый источник данных. Нажмите кнопку **OK**.

После выполнения этих дейстий имя `br` может быть использовано как DSN для базы данных **Борей**.

Объект *Recordset* и его создание

После установления соединения с источником данных можно создать объект `Recordset`. Этот объект представляет собой локальный кэш для исходной таблицы базы данных или результирующего набора считанных из источника записей, возвращаемых в результате запроса. Таким образом, объект `Recordset` это не реальный, а виртуальный набор записей из базы данных, позволяющий управлять данными в базе данных на уровне записи. На

уровне полей управление данными осуществляется объектом Field, причем свойство Fields возвращает семейство всех полей данного набора записей. Средства ADO позволяют создавать объект Recordset непосредственно или используя объекты Connection и Command. Объекты Recordset поддерживают четыре типа курсоров, перечисленных в табл. 15.2.

Таблица 15.2. *Типы курсоров объекта* Recordset

Тип	Описание
Dynamic (Динамический)	Позволяет видеть изменения в данных, сделанные другими пользователями, и поддерживает все типы движений, которые не основываются на закладках. Может разрешать использование закладок, если они поддерживаются поставщиком данных
Keyset (Управляемые ключи)	Является динамическим курсором, но не позволяет видеть записи, добавленные или удаленные другими пользователями. В то же время позволяет видеть изменения в данных, сделанные другими пользователяими. Поддерживает закладки и все виды перемещения по записям
Static (Статический)	Обеспечивает фиксированную копию данных все время, пока открыт набор записей. Дополнения, изменения и удаления, выполняемые другими пользователями, не имеют никакого воздействия на набор записей, т. к. он не обновляется после создания. Поддерживает закладки и все виды перемещения по записям
Forward-only (Однонаправленный)	Статический режим с последовательным доступом. Перемещаться от записи к записи можно только вперед. Это самый скоростной режим

Объект Recordset открывается методом Open. При этом надо указать таблицу или запрос, на основе которого строится этот объект, либо в параметре метода Open, либо при помощи свойства Source. Также требуется указать при помощи свойства ActiveConnection или одноименного параметра метода Open, на основе какого соединения конструируется объект Recordset.

Использование свойств *BOF* и *EOF* при навигации по записям

Свойства BOF и EOF при навигации по записям позволяют определить, достигнуто ли начало или конец набора записей:

- свойство EOF (End of File) устанавливается равным значению True при выходе за пределы последней записи;

- свойство BOF (Beginning of File) устанавливается равным значению True при переходе к позиции, находящейся перед первой записью.

При навигации по набору записей обычно используется следующий код:

```
Do Until rs.EOF
' Операторы
   rs.MoveNext
Loop
```

Свойства BOF и EOF всегда доступны, даже если набор записей пуст. Поэтому их можно использовать для проверки пустоты набора записей, а именно если они оба возвращают значение True, то набор пуст.

Пример использования объекта *Recordset* для просмотра базы данных

Приводимый проект на примере таблицы **Клиенты** базы данных **Борей** (рис. 15.1) демонстрирует, как можно создать объект Recordset. Итак, сконструируйте форму, на которой расположите список, а в модуле формы наберите следующий код (листинг 15.3). При заполнении списка надо последовательно перемещаться по записям объекта Recordset, что в коде производится методом MoveNext. Свойство EOF проверяет, находится ли указатель непосредственно после последней записи объекта Recordset.

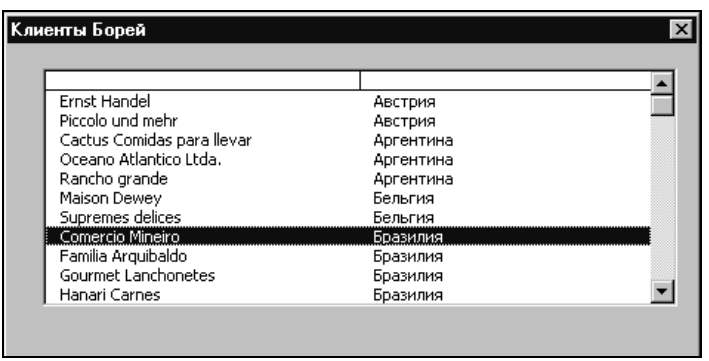


Рис. 15.1. Поля Название и Страна таблицы **Клиенты** базы данных **Борей**

Листинг 15.3. Создание объекта *Recordset*

```
Private cn As ADODB.Connection

Private Sub UserForm_Initialize()
   Me.Caption = "Клиенты Борей"
   ListBox1.ColumnCount = 2
```

```
   Set cn = New ADODB.Connection
   cn.Provider = "Microsoft.Jet.OLEDB.4.0"
   cn.ConnectionString = "c:\Борей.mdb"
   cn.Open

   Dim rs As New ADODB.Recordset
   rs.Source = "SELECT Название, Страна FROM Клиенты " & _
               "Order By Страна, Название"
   Set rs.ActiveConnection = cn
   rs.Open
   ListBox1.Clear
   ListBox1.ColumnHeads = True
   Dim i As Long
   i = 0
   Do Until rs.EOF
      ListBox1.AddItem rs.Fields("Название").Value
      ListBox1.List(i, 1) = rs.Fields("Страна").Value
      rs.MoveNext
      i = i + 1
   Loop
   rs.Close
End Sub

Private Sub UserForm_Terminate()
   cn.Close
   Set cn = Nothing
End Sub
```

Создание объекта *Recordset* без предварительного создания объекта *Connection*

Объект `Recordset` можно создать без использования объекта `Connection`. Для этого в свойстве `ActiveConnection` надо указать не ссылку на объект `Connection`, а строку, идентифицирующую источник данных (листинг 15.4).

Листинг 15.4. Создание объекта `Recordset` без предварительного создания объекта `Connection`

```
Private Sub UserForm_Initialize()
   ListBox1.ColumnCount = 2

   Dim rs As New ADODB.Recordset
   rs.Source = "SELECT Название, Страна FROM Клиенты " & _
               "Order By Страна, Название"
```

```
   rs.ActiveConnection = "DSN=br"
   rs.Open
   ListBox1.Clear
   ListBox1.ColumnHeads = True
   Dim i As Long
   i = 0
   Do Until rs.EOF
      ListBox1.AddItem rs.Fields("Название").Value
      ListBox1.List(i, 1) = rs.Fields("Страна").Value
      rs.MoveNext
      i = i + 1
   Loop
   rs.Close
End Sub
```

Методы, свойства и события объекта *Recordset*

Объект `Recordset` обладает обширной коллекцией методов, свойств и событий (табл. 15.3—15.5), позволяющих редактировать, создавать, обновлять записи, а также осуществлять полный контроль над этими и другими операциями.

Таблица 15.3. *Методы объекта* `Recordset`

Метод	Описание
`AddNew`	Подготавливает новую запись для добавления в набор записей. После внесения изменений в новую запись следует вызвать метод `Update` для сохранения изменений и добавления записи в объект `Recordset`. До вызова метода `Update` изменения в базу данных не заносятся
`Cancel`	Отмена асинхронного метода `Open`
`CancelBatch`	Отмена всех непринятых изменений в наборе записей
`CancelUpdate`	Отмена результатов операций `AddNew` и `Edit` без сохранения изменений
`Clone`	Создает копию набора записей
`Close`	Закрывает набор записей
`CompareBookmarks`	Определяет, какая из двух закладок указывает на предыдущую строку набора записей
`Delete`	Удаляет текущую запись
`Find`	Находит набор записей, удовлетворяющий специфицированному критерию

Таблица 15.3 *(окончание)*

Метод	Описание
`GetRows`	Заполняет массив записями из набора записей
`GetString`	Возвращает набор записей в виде строк с разделителями
`Move`	Перемещение из текущей записи к начальной. Можно указать число записей для перемещения
`MoveFirst`	Переход к первой записи
`MoveLast`	Переход к последней записи
`MoveNext`	Переход к следующей записи
`MovePrevious`	Переход к предыдущей записи
`NextRecordset`	Возвращение следующего набора записей из объекта `Connection`
`Open`	Открытие набора записей
`Requery`	Перезапуск исходного запроса, создавшего данный набор записей
`Resync`	Синхронизация набора записей с соответствующими ему данными
`Save`	Сохранение набора записей в файл
`Seek`	Поиск записи с использованием индекса
`Supports`	Определяет, какие операции поддерживает набор записей
`Update`	Подтверждает работу, выполненную методами `AddNew` и `Edit`
`UpdateBatch`	Подтверждает все текущие изменения в групповом курсоре

Таблица 15.4. *Свойства объекта* `Recordset`

Свойство	Описание
`AbsolutePage`	Количество страниц данных в текущей записи из набора записей
`AbsolutePosition`	Порядковый номер текущей записи в наборе записей
`ActiveCommand`	Команда, создавшая данный набор записей
`ActiveConnection`	Подключение, при помощи которого был создан данный набор записей
`BOF`	Возвращает значение `True`, если указатель текущей записи находится перед первой записью набора записей, и значение `False`, если указатель текущей записи расположен на первой записи набора или на любой записи после нее

Таблица 15.4 *(продолжение)*

Свойство	Описание
`Bookmark`	Закладка, уникальный идентификатор текущей записи набора записей
`CacheSize`	Число записей, кэшированных в оперативной памяти
`CursorLocation`	Константа, указывающая на местоположение курсора. Допустимыми значениями являются следующие константы: `adUseClient` и `adUseServer`
`CursorType`	Тип курсора. Допустимыми значениями являются следующие константы: `adOpenDynamic`, `adOpenForwardOnly`, `adOpenKeyset` и `adOpenStatic`
`DataMember`	Указывает принадлежность к компоненту набора записей при использовании связанных наборов данных
`DataSource`	Специфицирует источник данных, используемый для связанного набора данных
`EditMode`	Определяет, была ли отредактирована текущая запись. Допустимыми значениями являются следующие константы: `adEditAdd`, `adEditDelete`, `adEditInProgress` и `adEditNone`
`EOF`	Возвращает значение `True`, если указатель текущей записи находится после последней записи набора, и значение `False`, если указатель текущей записи расположен на последней записи набора или на любой записи перед ней
`Fields`	Семейство полей
`Filter`	Выбор по критерию поднабора записей из данного набора
`Index`	Определяет индекс, используемый при выполнении метода `Seek`
`LockType`	Задает тип блокировки, используемый при редактировании записей. Допустимыми значениями являются следующие константы: `adLockBatchOptimistic`, `adLockOptimistic`, `adLockPessimistic` и `adLockReadOnly`
`MarshalOptions`	Управляет маршализацией данных на сервере для набора записей на клиентской стороне
`MaxRecords`	Задает максимальное число записей, которые можно возвратить в набор записей
`PageCount`	Число страниц в наборе записей
`PageSize`	Размер страницы набора записей
`Properties`	Возвращает семейство `Properties`, описывающее набор записей

Таблица 15.4 *(окончание)*

Свойство	Описание
`RecordCount`	Число записей в наборе. Если ADO не может его определить, то возвращается значение –1
`Sort`	Задает поле для сортировки набора записей
`Source`	Команда или запрос SQL, создавший данный набор записей
`Status`	Результат операции группового обновления данных
`NoMatch`	Возвращаемое значение `True`, если нужная запись не найдена, и `False` — в противном случае
`StayInSync`	Применяется при обновлении иерархического набора

Таблица 15.5. *События объекта* `Recordset`

Событие	Описание
`EndOfRecordset`	Происходит при сбое метода `MoveNext` по причине отсутствия последующей записи
`FetchComplete`	Генерируется при достижении записей при помощи асинхронной операции
`FetchProgress`	Генерируется при выполнении асинхронной операции
`FieldChangeComplete`	Происходит после изменения значения поля
`MoveComplete`	Генерируется после перемещения указателя записи
`RecordChangeComplete`	Происходит после изменения целой записи
`RecordsetChangeComplete`	Генерируется после изменения во всем наборе записей
`WillChangeField`	Происходит перед изменением значения поля
`WillChangeRecord`	Происходит перед изменением целой записи
`WillChangeRecordset`	Происходит перед изменением во всем наборе записей
`WillMove`	Происходит перед перемещением указателя записи

Определение числа элементов в наборе записей

Для определения числа записей в наборе записей надо воспользоваться свойством `RecordCount` объекта `Recordset`, которое и возвращает искомое число.

Последовательный просмотр записей базы данных

Перемещаться по элементам набора записей можно при помощи методов `MoveFirst`, `MoveLast`, `MoveNext` и `MovePrevious` объекта `Recordset`. Продемонстрируем способ навигации по записям базы данных на примере таблицы **Клиенты** базы данных **Борей**. Создайте форму, на которой расположите две надписи, два поля ввода и четыре кнопки (рис. 15.2). Кнопки реализуют следующие функции:

- кнопка << — переход к первой записи;
- кнопка < — переход к предыдущей записи;
- кнопка > — переход к следущей записи;
- кнопка >> — переход к последней записи.

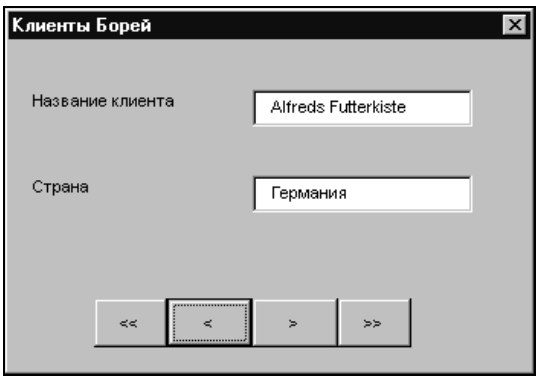


Рис. 15.2. Последовательный просмотр записей базы данных

При помощи окна **Properties** установите значения свойств элементов управления, как показано в табл. 15.6.

Таблица 15.6. *Значения свойств, установленные в окне* ***Properties***

Элемент управления	Свойство	Значение
Надпись	`Caption`	`Название`
Поле ввода	`Name`	`txtClientName`
Надпись	`Caption`	`Страна`
Поле ввода	`Name`	`txtCountry`
Кнопка	`Name`	`cmdFirst`
	`Caption`	`<<`

Таблица 15.6 (окончание)

Элемент управления	Свойство	Значение
Кнопка	Name	cmdPrevious
	Caption	<
Кнопка	Name	cmdNext
	Caption	>
Кнопка	Name	cmdLast
	Caption	>>

Установите ссылку на библиотеку доступа к данным. С этой целью в редакторе Visual Basic выберите команду **Tools | References** и убедитесь, что в списке **Available References** установлен флажок **Microsoft ActiveX Data Objects 2.0 Library**. Для завершения проекта осталось только в модуле формы набрать следующий код (листинг 15.5).

Листинг 15.5. Последовательный просмотр записей базы данных

```
Private rs As New ADODB.Recordset

Private Sub UserForm_Initialize()
   Me.Caption = "Клиенты Борей"
   rs.CursorType = adOpenKeyset
   rs.Source = "SELECT Название, Страна FROM Клиенты Order By Название"
   rs.ActiveConnection = "DSN=br"
   rs.Open
   cmdFirst_Click
End Sub

Private Sub cmdFirst_Click()
   rs.MoveFirst
   ShowRecord
End Sub

Private Sub cmdPrevious_Click()
   If Not rs.BOF Then
      rs.MovePrevious
      If Not rs.BOF Then
        ShowRecord
```

```
      Else
         rs.MoveFirst
      End If
   End If
End Sub

Private Sub cmdNext_Click()
   If Not rs.EOF Then
      rs.MoveNext
      If Not rs.EOF Then
         ShowRecord
      Else
         rs.MoveLast
      End If
   End If
End Sub

Private Sub cmdLast_Click()
   rs.MoveLast
   ShowRecord
End Sub

Private Sub ShowRecord()
   txtClientName.Text = rs.Fields("Название").Value
   txtCountry.Text = rs.Fields("Страна").Value
End Sub

Private Sub UserForm_Terminate()
   rs.Close
   Set rs = Nothing
End Sub
```

Восстановление объекта *Recordset* с помощью методов *Close* и *Open*

Объекты `Recordset`, объявленные на уровне модуля, могут использоваться повторно. После закрытия этого объекта методом `Close` можно заново установить свойства объекта, а затем повторно его открыть с помощью метода `Open`. Чтобы определить, нужно ли применять метод `Close` перед тем, как открыть объект с помощью метода `Open`, используется свойство `State`, которое возвращает значение `adStateOpen`, если набор записей открыт, и значение `adStateClosed`, если он закрыт.

```
If Not rs Is Nothing Then
   If rs.State = adStateOpen Then
      rs.Close
   End If
   Set rs = Nothing
End If
```

Работа с курсорами

ADO предусматривает поддержку нескольких типов курсоров. Помимо обеспечения последовательного перемещения по набору записей, курсоры позволяют контролировать способ управления набором записей.

Расположение курсора задается свойством `CursorLocation`, у которого имеются следующие два допустимых значения:

- `adUseServer` — курсор создается на стороне сервера;
- `adUseClient` — курсор создается на стороне клиента. Операциями с курсором управляют ADO и OLE DB. По сравнению с серверным курсором допустимо, например, создание отключенного набора записей, позволяющего манипулировать записями без постоянного подключения к серверу.

Свойство `CursorType` устанавливает тип курсора. Допустимыми значениями являются следующие константы:

- `adOpenDynamic` — динамический тип. Отслеживаются все изменения записей, произведенные другими пользователями;
- `adOpenForwardOnly` — однонаправленный тип. Перемещаться можно только вперед;
- `adOpenKeyset` — ключевой тип. Нельзя просматривать записи, добавленные в набор другими пользователями, но можно отслеживать сделанные ими обновления и удаления;
- `adOpenStatic` — статический. Работает с копией набора данных.

Объект *Field*

Объект `Field` представляет собой поле набора записей. Все поля образуют семейство `Fields`. У этого объекта имеются всего два метода и свойства, которые перечислены в табл. 15.7 и 15.8.

Таблица 15.7. *Методы объекта* `Field`

Метод	Описание
`AppendChunk`	Сохраняет порцию данных в поле двоичного числа типа `Long`
`GetChunk`	Изменяет порцию данных из поля двоичного числа типа `Long`

Таблица 15.8. *Свойства объекта* `Field`

Свойство	Описание
`ActualSize`	Размер данных, сохраненных в поле
`Attributes`	Список атрибутов поля
`DataFormat`	Управляет форматированием данных
`DefinedSize`	Максимальные размеры поля
`Name`	Имя поля
`NumericScale`	Разрядность числового поля
`OriginalValue`	Значение в поле перед тем, как его изменил пользователь
`Precision`	Число значащих цифр числового поля
`Properties`	Семейство `Properties`, описывающее свойства поля
`Type`	Тип данных поля
`UnderlyingValue`	Текущее значение, хранимое в базе данных для данного поля
`Value`	Текущее значение поля набора записей

Как при вводе данных в поле ввода из пустой ячейки таблицы избежать генерации ошибки

Если ячейка таблицы базы данных пуста, то при ее считывании возвращается значение `Null`, которое имеет тип, отличный от типа значения, вводимого в поле ввода — `String`. Для того чтобы предотвратить генерацию ошибки, принято использовать простой трюк — добавление пустой строки к возвращаемому из ячейки значению. Результирующее значение теперь всегда является строкой, а вместо значения `Null` в поле ввода вводится пустая строка.

Листинг 15.6. Предотвращение генерации ошибки при вводе данных в поле ввода из пустой ячейки

```
Private Sub ShowRecord()
   txtClientName.Text = rs.Fields("Название").Value & ""
   txtCountry.Text = rs.Fields("Страна").Value & ""
End Sub
```

Создание браузера базы данных

Для таблицы **Клиенты** базы данных **Борей** создадим простейший браузер. В этом браузере в список будут выводиться названия клиентов и их коды, причем последние нужны только для идентификации выбранной записи и поэтому выводятся в скрытый столбец списка. При выборе элемента из списка в полях ввода отображаются город и страна клиента, которые как раз и идентифицируются по скрытому в списке коду клиента (рис. 15.3).

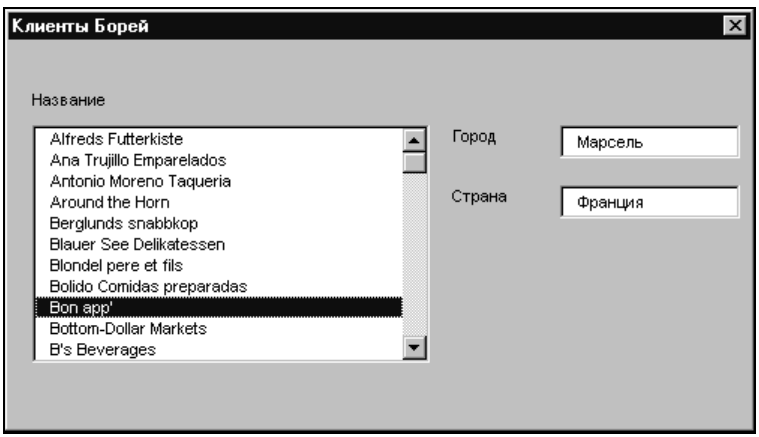


Рис. 15.3. Создание браузера базы данных

Итак, создайте форму, на которой расположите список, три надписи и два поля ввода и при помощи окна **Properties** установите им значения свойств, как показано в табл. 15.9.

Таблица 15.9. *Значения свойств, установленные в окне* ***Properties***

Элемент управления	Свойство	Значение
Надпись	Caption	Название
Список	Name	lstClient

Таблица 15.9 *(окончание)*

Элемент управления	Свойство	Значение
Надпись	Caption	Город
Поле ввода	Name	txtTown
Надпись	Caption	Страна
Поле ввода	Name	txtCountry

Установите ссылку на библиотеку доступа к данным. С этой целью в редакторе Visual Basic выберите команду **Tools | References** и убедитесь, что в списке **Available References** установлен флажок **Microsoft ActiveX Data Objects 2.0 Library**. Для завершения проекта осталось только в модуле формы набрать следующий код (листинг 15.7).

Листинг 15.7. Браузер базы данных

```
Private rs As New ADODB.Recordset

Private Sub UserForm_Initialize()
   Me.Caption = "Клиенты Борей"
   Dim idx As Long
   lstClient.ColumnCount = 2
   lstClient.ColumnWidths = "80;0"
   rs.Source = _
      "SELECT КодКлиента, Название FROM Клиенты Order By Название"
   rs.ActiveConnection = "DSN=br"
   rs.Open
   idx = 0
   Do Until rs.EOF
      lstClient.AddItem rs.Fields("Название")
      lstClient.List(idx, 1) = rs.Fields("КодКлиента")
      idx = idx + 1
      rs.MoveNext
   Loop
   rs.Close
   lstClient.ListIndex = 0
End Sub

Private Sub lstClient_Click()
   Dim id As String
   id = lstClient.List(lstClient.ListIndex, 1)
```

```
   rs.Source = "SELECT Город, Страна FROM Клиенты WHERE КодКлиента=" _
               & Chr(39) & id & Chr(39)
   rs.Open
   txtTown.Text = rs.Fields("Город").Value
   txtCountry.Text = rs.Fields("Страна").Value
   rs.Close
End Sub

Private Sub UserForm_Terminate()
   Set rs = Nothing
End Sub
```

Браузер просмотра нескольких таблиц базы данных

В *разд. "Последовательный просмотр записей базы данных"* приводился пример браузера просмотра одной таблицы базы данных. Настало время продемонстрировать технику работы с базой данных, состоящей из нескольких таблиц. Сделаем это на примере учебной базы данных **Борей**. Создайте форму, на которой расположите два списка и две надписи (рис. 15.4).

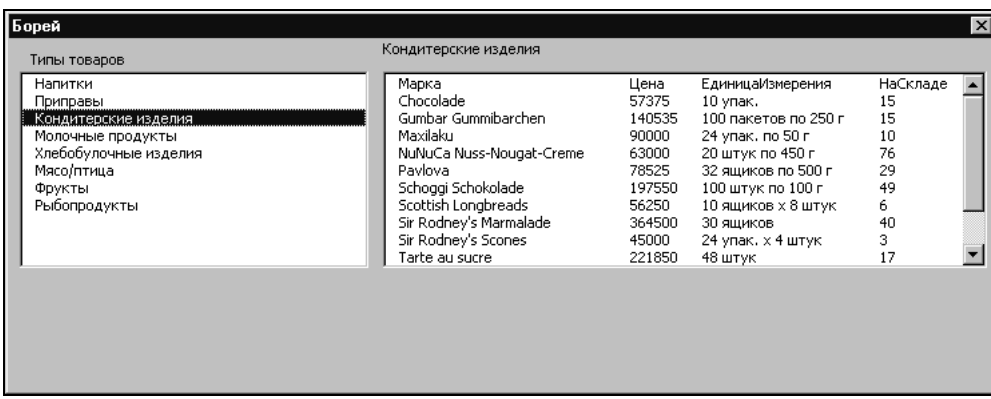


Рис. 15.4. Браузер базы данных **Борей**

В левый список при инициализации формы выведем все записи из поля `Категория` таблицы **Типы**. Кроме того, в скрытый столбец списка выведем соответствующие значения из поля `КодТипа`, которое является первичным ключом этой таблицы.

При выборе категории товара из левого списка в надпись, расположенную над правым списком, выводится указанная категория товара. Создается запрос к таблице **Товары**. По этому запросу выбираются все записи из полей

Марка, Цена, ЕдиницаИзмерения, НаСкладе, такие, что значение поля КодТипа (которое является внешним ключом) совпадает с указанным значением поля КодТипа из таблицы **Типы**. Результат запроса выводится во второй список.

Для реализации проекта при помощи окна **Properties** установите значения свойств элементов управления, как показано в табл. 15.10.

Таблица 15.10. *Значения свойств, установленные в окне* ***Properties***

Элемент управления	Свойство	Значение
Надпись	Name	lblCategory
	Caption	Типы товаров
Список	Name	lstCategory
Надпись	Name	lblGood
Список	Name	lstGood

Установите ссылку на библиотеку доступа к данным. С этой целью в редакторе Visual Basic выберите команду **Tools | References** и убедитесь, что в списке **Available References** установлен флажок **Microsoft ActiveX Data Objects 2.0 Library**. Для завершения проекта осталось только в модуле формы набрать следующий код (листинг 15.8).

Листинг 15.8. Еще один браузер базы данных

```
Private rs As New ADODB.Recordset

Private Sub UserForm_Initialize()
   With lstCategory
      .ColumnCount = 2
      .ColumnWidths = "120;0"
   End With
   With lstGood
      .ColumnCount = 4
      .ColumnWidths = "130;40;100;40"
   End With

   Dim sqlQuery As String
   Dim i As Long
   Me.Caption = "Борей"
   rs.CursorType = adOpenKeyset
   rs.LockType = adLockOptimistic
   sqlQuery = "SELECT КодТипа, Категория FROM Типы"
```

```
   rs.Source = sqlQuery
   rs.ActiveConnection = "DSN=br"
   rs.Open
   Application.Cursor = xlWait
   i = 0
   Do Until rs.EOF
      lstCategory.AddItem rs.Fields("Категория")
      lstCategory.List(i, 1) = rs.Fields("КодТипа")
      i = i + 1
      rs.MoveNext
   Loop
   rs.Close
   Application.Cursor = xlDefault
   lstCategory.ListIndex = 0
End Sub

Private Sub lstCategory_Click()
   Dim sqlQuery As String
   Dim idCategory As Long
   If lstCategory.ListIndex = -1 Then Exit Sub
   lblGood.Caption = lstCategory.Text
   idCategory = lstCategory.List(lstCategory.ListIndex, 1)
   sqlQuery = "SELECT Марка, Цена, ЕдиницаИзмерения, НаСкладе " _
              & "FROM Товары WHERE КодТипа = " _
              & CStr(idCategory) & " ORDER BY Марка"
   Application.Cursor = xlWait
   Debug.Print sqlQuery
   rs.Source = sqlQuery
   rs.ActiveConnection = "DSN=br"
   rs.Open
   With lstGood
      .Clear
      .AddItem rs.Fields("Марка").Name
      .List(0, 1) = rs.Fields("Цена").Name
      .List(0, 2) = rs.Fields("ЕдиницаИзмерения").Name
      .List(0, 3) = rs.Fields("НаСкладе").Name
   End With

   Dim i As Long
   i = 1
   Do Until rs.EOF
      With lstGood
         .AddItem rs.Fields("Марка").Value
         .List(i, 1) = rs.Fields("Цена").Value
```

```
            .List(i, 2) = rs.Fields("ЕдиницаИзмерения").Value
            .List(i, 3) = rs.Fields("НаСкладе").Value
        End With
        i = i + 1
        rs.MoveNext
    Loop
    rs.Close
    Application.Cursor = xlDefault
End Sub
```

Сохранение и открытие набора записей файлового типа

Набор записей можно сохранять в RST-файл методом `Save`. Для открытия сохраненного файла надо указать в качестве значения свойства `Source` полное имя этого файла и установить для параметра `Options` метода `Open` значение `adCmdFile`. Следующий проект демонстрирует, как можно сохранять и открывать набор записей файлового типа. Создайте форму, на которой расположите список и две кнопки (рис. 15.5), и при помощи окна **Properties** установите значения свойств элементов управления, как показано в табл. 15.11.

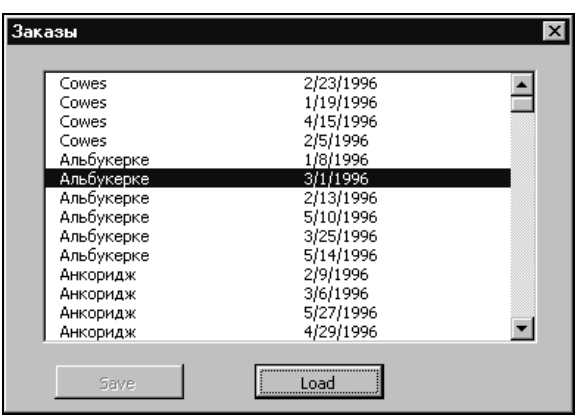


Рис. 15.5. Сохранение и открытие набора записей файлового типа

Таблица 15.11. *Значения свойств, установленные в окне* ***Properties***

Элемент управления	Свойство	Значение
Кнопка	`Name`	`cmdSave`
	`Caption`	`Save`

Таблица 15.11 (окончание)

Элемент управления	Свойство	Значение
Кнопка	Name	cmdLoad
	Caption	Load

Установите ссылку на библиотеку доступа к данным. С этой целью в редакторе Visual Basic выберите команду **Tools | References** и убедитесь, что в списке **Available References** установлен флажок **Microsoft ActiveX Data Objects 2.0 Library**. Для завершения проекта осталось только в модуле формы набрать следующий код (листинг 15.9).

Листинг 15.9. Сохранение и открытие набора записей файлового типа

```
Private rs As New ADODB.Recordset

Private Sub UserForm_Initialize()
   ListBox1.ColumnCount = 2
   Dim sqlQuery As String
   sqlQuery = "SELECT * FROM Заказы " & _
              "WHERE ДатаИсполнения " & _
              "BETWEEN #1/1/96# AND #12/31/96# ORDER BY ГородПолучателя"
   rs.ActiveConnection = "DSN=br"
   rs.CursorType = adOpenKeyset
   rs.Source = sqlQuery
   rs.Open
   ListBox1.Clear
   Dim i As Long
   i = 0
   Do Until rs.EOF
      ListBox1.AddItem rs.Fields("ГородПолучателя").Value
      ListBox1.List(i, 1) = rs.Fields("ДатаИсполнения").Value
      rs.MoveNext
      i = i + 1
   Loop
   cmdSave.Enabled = True
   cmdLoad.Enabled = False
End Sub

Private Sub cmdSave_Click()
   If Len(Dir(ThisWorkbook.Path & "\Orders.rst")) > 0 Then
      Kill ThisWorkbook.Path & "\Orders.rst"
   End If
```

```
   rs.Save ThisWorkbook.Path & "\Orders.rst"
   ListBox1.Clear
   rs.Close
   Set rs = Nothing
   cmdSave.Enabled = False
   cmdLoad.Enabled = True
End Sub

Private Sub cmdLoad_Click()
   Dim rs As New ADODB.Recordset
   rs.Source = ThisWorkbook.Path & "\Orders.rst"
   rs.CursorType = adOpenStatic
   rs.LockType = adLockPessimistic
   rs.Open Options:=adCmdFile
   Application.Cursor = xlWait
   Dim i As Long
   i = 0
   Do Until rs.EOF
      ListBox1.AddItem rs.Fields("ГородПолучателя").Value
      ListBox1.List(i, 1) = rs.Fields("ДатаИсполнения").Value
      rs.MoveNext
      i = i + 1
   Loop
   Application.Cursor = xlDefault
   rs.Close
   Set rs = Nothing
End Sub
```

Редактирование, создание, обновление и удаление записей

При создании и открытии набора записей значения нескольких свойств, определяющих этот набор, устанавливаются по умолчанию. Если же требуется внести в набор какие-либо изменения, то значения этих свойств, а именно `LockType` и `CursorType`, также надо изменить. Свойство `LockType` может принимать следующие значения:

- `adLockBatchOptimistic` — разрешено пакетное обновление;
- `adLockOptimistic` — устанавливает блокировку отдельных записей, причем запись блокируется при вызове метода `Update`;
- `adLockPessimistic` — устанавливает блокировку отдельных записей, причем запись блокируется сразу же после начала редактирования;

- `adLockReadOnly` — объект `Recordset` доступен "только для чтения". Редактирование запрещено. Это значение устанавливается по умолчанию.

По умолчанию набор записей открывается со значением свойства `LockType`, равным `adLockReadOnly`. Чтобы обойти это ограничение, надо установить его значение равным либо `adLockOptimistic`, либо `adLockPessimistic`. Различие между этими двумя значениями проявляется только при работе в многопользовательском режиме и состоит в том, что записи блокируются либо при обновлении, либо при редактировании.

Обновление записи производится методом `Update`, создание новой записи — методом `AddNew`, а удаление текущей записи — методом `Delete`.

Продемонстрируем работу этих методов на примере табличной базы данных **Phone**, состоящей из единственной таблицы `tblPhone` (DSN которой установлен равным `ph`) с тремя полями:

```
Фамилия
Имя
Телефон
```

Итак, создайте форму, на которой расположите семь кнопок, три надписи и три поля ввода (рис. 15.6). Установите им при помощи окна **Properties** значения свойств, как показано в табл. 15.12.

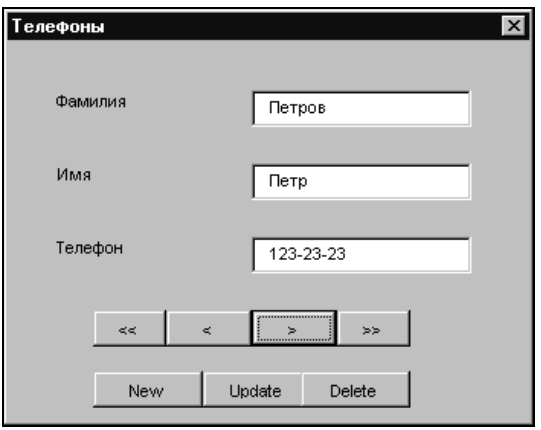


Рис. 15.6. Редактирование, создание, обновление и удаление записей

Таблица 15.12. *Значения свойств, установленные в окне* ***Properties***

Элемент управления	Свойство	Значение
Надпись	`Caption`	`Название`
Поле ввода	`Name`	`txtClientName`

Таблица 15.12 *(окончание)*

Элемент управления	Свойство	Значение
Надпись	Caption	Страна
Поле ввода	Name	txtCountry
Кнопка	Name	cmdFirst
	Caption	<<
Кнопка	Name	cmdPrevious
	Caption	<
Кнопка	Name	cmdNext
	Caption	>
Кнопка	Name	cmdLast
	Caption	>>
Кнопка	Name	cmdNew
	Caption	New
Кнопка	Name	cmdUpdate
	Caption	Update
Кнопка	Name	cmdDelete
	Caption	Delete

Кнопки <<, <, > и >> будут осуществлять навигацию по таблице базы данных. Остальные кнопки выполняют следующие функции:

- кнопка **Update** предназначена для обновления записи в базе данных;
- кнопка **Delete** предназначена для удаления записи из базы данных;
- кнопка **New** предназначена для создания новой записи. При этом при нажатии кнопки **New** все кнопки окна, за исключением кнопки **Update**, блокируются для пользователя. Таким образом, пользователь не может осуществлять никаких операций над записями до тех пор, пока не запишет новую запись в базу данных. При нажатии же кнопки **Update** происходит разблокирование всех кнопок.

В приводимом ниже коде (см. листинг 15.10) за процесс управления блокировкой кнопок отвечает процедура `SetEnabled`, а переменная `IsDisable` специфицирует достижимость кнопок для пользователя.

Не забудьте установить ссылку на библиотеку доступа к данным. С этой целью в редакторе Visual Basic выберите команду **Tools | References** и убедитесь, что в списке **Available References** установлен флажок **Microsoft ActiveX**

Data Objects 2.0 Library. Для завершения проекта осталось только в модуле формы набрать следующий код (листинг 15.10).

Обратите внимание, что использование свойства `Tag`, а также префиксов в именах элементов управления, позволяет идентифицировать требуемый элемент управления (кнопку) при переборе в цикле

```
For Each ctrl In Controls
Next
```

Идентификация же кнопки **Update** в этом цикле производится по значению свойства `Tag`.

Листинг 15.10. Редактирование, создание и удаление записей

```
Private rs As New ADODB.Recordset

Private IsDisable As Boolean

Private Sub UserForm_Initialize()
   Me.Caption = "Телефоны"
   rs.CursorType = adOpenKeyset
   rs.LockType = adLockOptimistic
   rs.Source = "SELECT * FROM tblPhone ORDER BY Фамилия"
   rs.ActiveConnection = "DSN=ph"
   rs.Open
   cmdUpdate.Tag = "Update"
   cmdFirst_Click
   IsDisable = False
End Sub

Private Sub cmdFirst_Click()
   rs.MoveFirst
   ShowRecord
End Sub

Private Sub cmdPrevious_Click()
   If Not rs.BOF Then
      rs.MovePrevious
      If Not rs.BOF Then
        ShowRecord
      Else
         rs.MoveFirst
      End If
   End If
End Sub
```

```
Private Sub cmdNext_Click()
   If Not rs.EOF Then
      rs.MoveNext
      If Not rs.EOF Then
         ShowRecord
      Else
         rs.MoveLast
      End If
   End If
End Sub

Private Sub cmdLast_Click()
   rs.MoveLast
   ShowRecord
End Sub

Private Sub cmdNew_Click()
   ShowEmptyRecord
   rs.AddNew
   FillRecord
   txtLName.SetFocus
   IsDisable = True
   SetEnabled True, False
End Sub

Private Sub cmdUpdate_Click()
   If Not IsDisable Then rs.Update
   FillRecord
   SetEnabled True, True
   If IsDisable Then
      rs.MoveLast
      IsDisable = False
   End If
End Sub

Private Sub cmdDelete_Click()
   If rs.RecordCount >= 1 Then
      If MsgBox("Удалить текущую запись?", vbYesNo + vbQuestion) _
          = vbYes Then
         rs.Delete
         If rs.RecordCount > 0 Then
            cmdNext_Click
         Else
            ShowEmptyRecord
         End If
```

```
      End If
   End If
End Sub

Private Sub SetEnabled(IsUpdateOn As Boolean, IsOthersOn As Boolean)
    Dim ctrl As Control
    For Each ctrl In Controls
       If LCase(Left(ctrl.Name, 3)) = "cmd" Then
          If ctrl.Tag = "Update" Then
             ctrl.Enabled = IsUpdateOn
          Else
             ctrl.Enabled = IsOthersOn
          End If
       End If
    Next
End Sub

Private Sub ShowRecord()
   txtLName.Text = rs.Fields("Фамилия").Value
   txtFName.Text = rs.Fields("Имя").Value
   txtPhone.Text = rs.Fields("Телефон").Value
End Sub

Private Sub FillRecord()
   rs.Fields("Фамилия").Value = txtLName.Text
   rs.Fields("Имя").Value = txtFName.Text
   rs.Fields("Телефон").Value = txtPhone.Text
End Sub

Private Sub ShowEmptyRecord()
   txtLName.Text = Empty
   txtFName.Text = Empty
   txtPhone.Text = Empty
End Sub

Private Sub UserForm_Terminate()
   rs.Close
   Set rs = Nothing
End Sub
```

Создание закладок

При работе с базами данных удобно устанавливать закладки у наиболее часто используемых записей и, используя эти закладки, быстро переходить к требуемым записям. Продемонстрируем, как это делается, на примере

проекта с базой данных **Phone** из предыдущего раздела. Добавьте на форму еще две кнопки (рис. 15.7) и установите им при помощи окна **Properties** значения свойств, как показано в табл. 15.13.

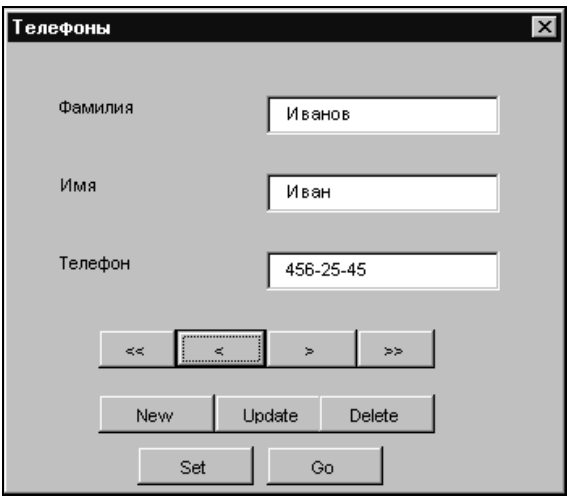


Рис. 15.7. Телефонная книга с закладками

Таблица 15.13. *Значения свойств, установленные в окне* ***Properties***

Элемент управления	Свойство	Значение
Кнопка	Name	cmdSet
	Caption	Set
Кнопка	Name	cmdGo
	Caption	Go

Для завершения проекта осталось только в модуль формы добавить следующий код (листинг 15.11).

Внимание!

В целях экономии места и во избежание рутинных повторений код процедур, обрабатывающих события нажатия кнопок "<<", "<", ">", ">>", **New**, **Update** и **Delete**, а также события `Initialize` и `Terminate` объекта `UserForm`, в приводимой программе опущен. Не забудьте его добавить в программу из листинга 15.10 предыдущего раздела.

Листинг 15.11. Установка закладки

```
Private bm As Variant

Private Sub cmdSetBookmark_Click()
   bm = rs.Bookmark
End Sub

Private Sub cmdGoToBookmark_Click()
   If bm <> Empty Then
      rs.Bookmark = bm
      ShowRecord
   Else
      MsgBox "Закладка не установлена"
   End If
End Sub
```

Поиск записи методом *Find*

В этом разделе мы изучим важную операцию — поиск записи по заданному критерию. Объект `Recordset` обладает методом `Find`, который позволяет производить подобный поиск записи по одному полю.

```
Find(Criteria As String, [SkipRecords As Long],
↳[SearchDirection As SearchDirectionEnum = adSearchForward], [Start])
```

- *`Criteria`* — критерий, по которому ведется поиск в наборе записей. Этот параметр не допускает общих SQL-запросов типа `WHERE`. Операторы сравнения в нем ограничиваются только следующим списком: =, >, >=, <, <= и `LIKE`;
- *`SkipRecords`* — задает число уже найденных пропущенных записей, после которых только начинается поиск;
- *`SearchDirection`* — определяет направление поиска. Допустимыми значениями являются константы `adSearchBackward` и `adSearchForward`;
- *`Start`* — позиция, с которой начинается поиск.

Для того чтобы найти первое вхождение искомой записи в набор записей, надо воспользоваться методом `Find` со значением параметра *`SkipRecords`*, равным 0, а параметра *`SearchDirection`* — равным `adSearchForward`.

Для того чтобы найти последующее вхождение искомой записи, применяется метод `Find` со значением параметра *`SkipRecords`*, равным 1, значением параметра *`SearchDirection`*, равным `adSearchForward`, а значение параметра *`Start`* должно давать ссылку на закладку последней найденной записи, удовлетворяющей данному критерию.

Продемонстрируем, как работает метод `Find` на примере проекта с базой данных **Phone** из предыдущего раздела. Добавьте на форму еще две кнопки и поле ввода (рис. 15.8) и установите им при помощи окна **Properties** значения свойств, как показано в табл. 15.14.

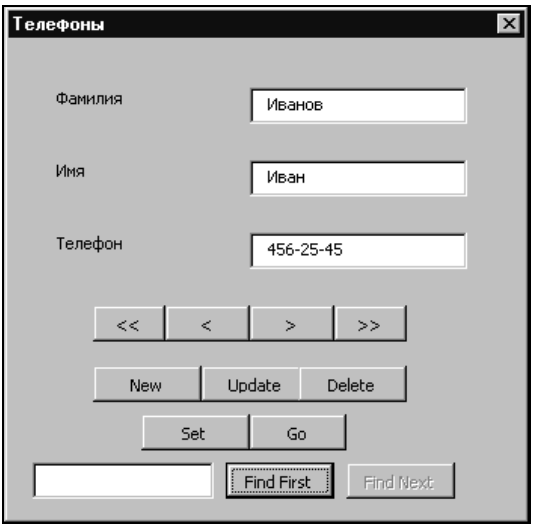


Рис. 15.8. Поиск в телефонной книге

Таблица 15.14. *Значения свойств, установленные в окне* ***Properties***

Элемент управления	Свойство	Значение
Кнопка	`Name`	`Find First`
	`Caption`	`Set`
Кнопка	`Name`	`cmdFindNext`
	`Caption`	`Find Next`
Поле ввода	`Name`	`txtFind`

Для завершения проекта осталось только в модуль формы добавить следующий код (листинг 15.12).

Внимание!

В целях экономии места и во избежание рутинных повторений код процедур, обрабатывающих события нажатия кнопок "<<", "<", ">", ">>", **New**, **Update**, **Delete**, **Set** и **Go**, а также события `Initialize` и `Terminate` объекта `UserForm`, в приводимой программе опущен. Не забудьте его добавить в программу из листингов 15.10 и 15.11 двух предыдущих разделов.

При нажатии кнопки **Find First** осуществляется поиск по фамилии, введенной в поле, расположенное рядом с этой кнопкой. Если таковой фамилии не имеется, то на экране отображается окно с соответствующим сообщением, а кнопка **Find Next** блокируется. Если же имеется, то отображается подтверждающее сообщение, искомая запись выводится в соответствующие поля, а кнопка **Find Next** разблокируется.

Нажатие кнопки **Find Next** производит поиск следующей записи, удовлетворяющей критерию поиска. В случае отсутствия таковой пользователь информируется, а кнопка **Find Next** блокируется.

Листинг 15.12. Поиск записи методом `Find`

```
' В процедуру обработки события Initialize объекта UserForm
' добавьте инструкцию
' cmdFindNext.Enabled = False
'
Private bmark As Variant
Private LName As String
Private criteria As String

Private Sub cmdFindFirst_Click()
   Dim skiprecord As Long
   Dim direction As Long
   LName = txtFind.Text
   criteria = "[Фамилия] = '" & LName & "'"
   skiprecord = 0
   direction = adSearchForward
   bmark = rs.Bookmark
   rs.MoveFirst
   rs.Find criteria, skiprecord, direction
   If rs.EOF Then
       MsgBox "no found"
       Beep
       rs.Bookmark = bmark
       cmdFindNext.Enabled = False
   Else
       MsgBox "found"
       Beep
       ShowRecord
       cmdFindNext.Enabled = True
   End If
End Sub
```

```
Private Sub cmdFindNext_Click()
   Dim skiprecord As Long
   Dim direction As Long
   bmark = rs.Bookmark
   skiprecord = 1
   direction = adSearchForward
   rs.Find criteria, skiprecord, direction, bmark
   If rs.EOF Then
       MsgBox "no found"
       Beep
       rs.Bookmark = bmark
       cmdFindNext.Enabled = False
   Else
       MsgBox "found"
       Beep
       ShowRecord
   End If
End Sub
```

Как избежать написания большого числа апострофов при построении критерия поиска

В качестве значения параметра `Criteria` методов группы `Find` нельзя использовать двойные кавычки. Таким образом, правильным критерием для поиска записи, у которой в поле **Фамилия** размещается значение "`Петров`", будет

```
"[Фамилия]= 'Петров'"
```

В случае если используется не конкретное имя "`Петров`", а то, которое считывается из поля ввода, например `txtLName`, как это было в рассмотренном в предыдущем разделе примере, критерий примет вид

```
"[Фамилия]='" & txtLName.Text & "'"
```

Если последнее выражение из-за чрезмерного количества апострофов представляется излишне неказистым, то воспользуйтесь следующим элегантным подходом. Создайте функцию `Quate`, которая окружает с обеих сторон строковое выражение кавычками:

```
Function Quate(s As String) As String
   AddQuate = Chr(34) & s & Chr(34)
End Function
```

И уж эту функцию примените при составлении критерия.

```
"[Фамилия]=" & Quate(txtFind.Text)
```

Применение баз данных при создании интернациональных приложений

Базы данных удобно применять при создании интернациональных приложений. Рассмотрим простейший пример на эту тему. В данном приложении при выборе соответствующего положения выключателя все надписи в окне будут переводиться либо на русский, либо на английский язык. Итак, для реализации этого приложения создайте форму, на ней разместите две надписи, список и выключатель (рис. 15.9), и установите им при помощи окна **Properties** значения свойств, как показано в табл. 15.15.

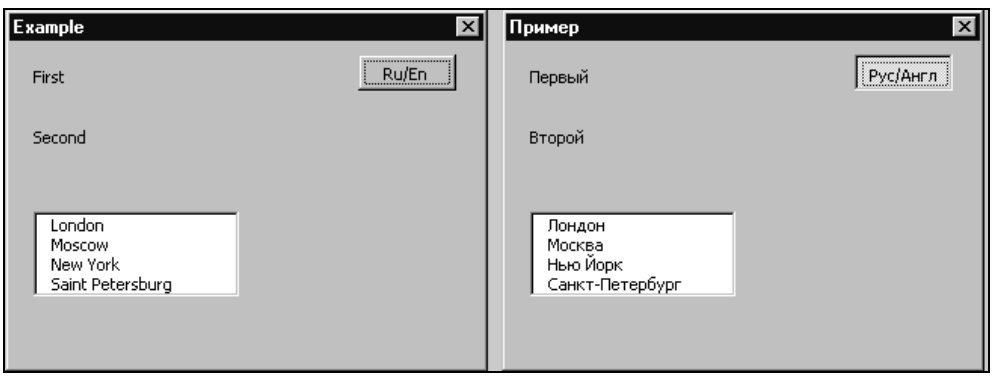


Рис. 15.9. Окно с переключателем языков

Таблица 15.15. *Значения свойств, установленные в окне* ***Properties***

Элемент управления	Свойство	Значение
Форма	Name	frmLanguage
Надпись		lblFirst
Надпись	Name	lblSecond
Выключатель		tglLanguage
Список	Name	lstCity
	Tag	No

С помощью MS Access создайте базу данных Language.mdb, состоящую из двух таблиц (рис. 15.10):

tblCity

Русский	Русский текст, отображаемый в списке
Английский	Английский текст, отображаемый в списке

tblLanguage

Объект	Значения свойства Name элементов управления и формы
Русский	Русские значения свойства Caption
Английский	Английские значения свойства Caption

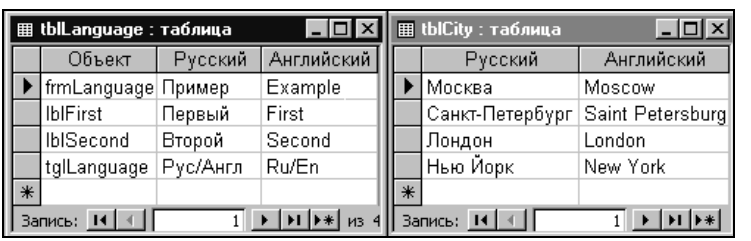


Рис. 15.10. Таблицы **tblCity** и **tblLanguage**

Задайте имя источника данных DSN (базу данных Language.mdb) равным `language`. Теперь все готово для работы, за исключением самой малости — набора в модуле формы следующего кода (листинг 15.13).

Листинг 15.13. Создание интернациональных приложений

```
Private Sub UserForm_Initialize()
   tglLanguage.Value = False
   ChangeCaptions
End Sub

Private Sub tglLanguage_Change()
   ChangeCaptions
End Sub

Private Sub ShowLocal(Language As String)
   Dim rs As New ADODB.Recordset
   Dim cnt As Control

   rs.CursorType = adOpenKeyset
   rs.Source = "SELECT * FROM tblLanguage"
```

```
   rs.ActiveConnection = "DSN=language"
   rs.Open
   Dim skiprecord As Long
   Dim direction As Long
   skiprecord = 0
   direction = adSearchForward
   Dim criteria As String
   Dim bmark As Variant

   For Each cnt In frmLanguage.Controls
      bmark = rs.Bookmark
      rs.MoveFirst
      If cnt.Tag <> "No" Then
         criteria = "[Объект]='" & cnt.Name & "'"
         rs.Find criteria, skiprecord, direction
         cnt.Caption = rs.Fields(Trim(Language)).Value
      End If
   Next

   bmark = rs.Bookmark
   rs.MoveFirst
   criteria = "[Объект]='" & frmLanguage.Name & "'"
   rs.Find criteria, skiprecord, direction
   frmLanguage.Caption = rs.Fields(Trim(Language)).Value
   rs.Close

   rs.Source = "SELECT * FROM tblCity ORDER BY Русский"
   rs.Open
   lstCity.Clear
   Do Until rs.EOF
      lstCity.AddItem rs.Fields(Trim(Language)).Value
      rs.MoveNext
   Loop
   rs.Close
   Set rs = Nothing
End Sub

Private Sub ChangeCaptions()
   If tglLanguage.Value Then
      ShowLocal "Русский"
   Else
      ShowLocal "Английский"
   End If
End Sub
```

Как сохранить набор записей в XML-формате

Начиная с ADO 2.5, появилась возможность сохранять набор записей в XML-формате. Этой операцией управляет метод `Save` объекта `Recordset`. Причем, достаточно в качестве значения его параметра `PersistFormat` указать константу `adPersistXML`. Следующий код (листинг 15.14) выводит таблицу `tblPhone` базы данных Phonebook.mdb в виде XML-файла Test.xml, а также в виде потока, который выводится в окно отладки.

Листинг 15.14. Сохранение набора записей в виде XML-файла

```
Public Function SaveToXML(rs As ADODB.Recordset) As String
   Dim adoStream As ADODB.stream
   On Error GoTo ErrorHandler
   Set adoStream = New ADODB.stream
   rs.Save Destination:=adoStream, PersistFormat:=adPersistXML
   rs.Save Destination:="test.xml", PersistFormat:=adPersistXML
   SaveToXML = adoStream.ReadText
   Set adoStream = Nothing
   Exit Function

ErrorHandler:
   MsgBox "Error"
End Function

Public Sub DemoSaveToXML()
   Dim rs As New ADODB.Recordset
   rs.CursorType = adOpenKeyset
   rs.LockType = adLockOptimistic
   rs.Source = "SELECT * FROM tblPhone ORDER BY Фамилия"
   rs.ActiveConnection = "DSN=ph"
   rs.Open
   Debug.Print SaveToXML(rs)
End Sub
```

В результате работы данной программы будет создан файл Test.xml следующего содержания:

```
<xml xmlns:s='uuid:BDC6E3F0-6DA3-11d1-A2A3-00AA00C14882'
   xmlns:dt='uuid:C2F41010-65B3-11d1-A29F-00AA00C14882'
   xmlns:rs='urn:schemas-microsoft-com:rowset'
   xmlns:z='#RowsetSchema'>
```

```
<s:Schema id='RowsetSchema'>
   <s:ElementType name='row' content='eltOnly' rs:updatable='true'>
      <s:AttributeType name='Фамилия' rs:number='1' rs:nullable='true'
          rs:write='true' rs:basetable='tblPhone'
          rs:basecolumn='Фамилия'>
         <s:datatype dt:type='string' dt:maxLength='50'/>
      </s:AttributeType>
      <s:AttributeType name='Имя' rs:number='2' rs:nullable='true'
          rs:write='true' rs:basetable='tblPhone' rs:basecolumn='Имя'>
         <s:datatype dt:type='string' dt:maxLength='50'/>
      </s:AttributeType>
      <s:AttributeType name='Телефон' rs:number='3' rs:nullable='true'
          rs:write='true' rs:basetable='tblPhone'
          rs:basecolumn='Телефон'>
         <s:datatype dt:type='string' dt:maxLength='50'/>
      </s:AttributeType>
      <s:extends type='rs:rowbase'/>
   </s:ElementType>
</s:Schema>
<rs:data>
   <z:row Фамилия='Иванов' Имя='Иван' Телефон='456-25-45'/>
   <z:row Фамилия='Никифоров' Имя='Никифор' Телефон='345-47-34'/>
   <z:row Фамилия='Петров' Имя='Петр' Телефон='345-47-34'/>
   <z:row Фамилия='Петров' Имя='Иван' Телефон='143-47-34'/>
   <z:row Фамилия='Петров' Имя='Федор' Телефон='123-23-23'/>
   <z:row Фамилия='Сидоров' Имя='Сидор' Телефон='123-65-24'/>
   <z:row Фамилия='Фомин' Имя='Фома' Телефон='123-12-31'/>
</rs:data>
</xml>
```

Вывод набора записей на рабочий лист

Для вывода набора записей на рабочий лист можно воспользоваться методом `CopyFromRecordset` объекта `Range`.

`CopyFromRecordset(`*`Data`*`, `*`MaxRows`*`, `*`MaxColumns`*`)`

- *`Data`* — обязательный параметр, задающий копируемый набор записей;
- *`MaxRows`* — необязательный параметр, определяющий максимальное число копируемых записей;
- *`MaxColumns`* — необязательный параметр, определяющий максимальное число копируемых полей.

В следующем примере (листинг 15.15) на рабочий лист книги выводятся записи из таблицы **Клиенты** базы данных **Борей**.

Листинг 15.15. Вывод набора записей на рабочий лист

```
Private Sub LoadRecordsetToWorkbook()
   Dim cn As New ADODB.Connection
   cn.Provider = "Microsoft.Jet.OLEDB.4.0"
   cn.ConnectionString = "c:\Борей.mdb"
   cn.Open

   Dim rs As New ADODB.Recordset
   rs.Source = "SELECT Название, Страна FROM Клиенты"
   Set rs.ActiveConnection = cn
   rs.Open
   Dim NewBook As Workbook
   Dim i As Integer
   Set NewBook = Workbooks.Add

   For i = 0 To rs.Fields.Count - 1
      NewBook.Sheets(1).Range("A1").Offset(0, i).Value _
         = rs.Fields(i).Name
   Next
   NewBook.Sheets(1).Range("A2").CopyFromRecordset rs
   rs.Close
   Set rs = Nothing
   cn.Close
   Set cn = Nothing
End Sub
```

Использование объекта *Command*

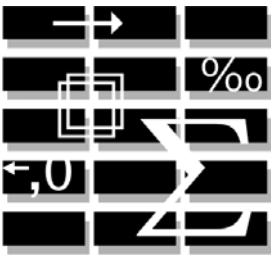
Объект `Command` позволяет применять SQL-команды. Основным свойством этого объекта является свойство `CommandText`, которое задает SQL-команду, пересылаемую в источник данных через поставщика OLE DB. В следующем коде (листинг 15.16), например, в базу данных Phonebook.mdb вводится очередная запись, причем в ней только специфицируются значения полей **Фамилия** и **Телефон**.

Листинг 15.16. Использование объекта `Command`

```
Sub InsertRecord()
   Set cn = New ADODB.Connection
```

```
    cn.Provider = "Microsoft.Jet.OLEDB.4.0"
    cn.ConnectionString = "C:\phonebook.mdb"
    cn.Open
    Dim cmd  As New ADODB.Command
    cmd.CommandText = "INSERT INTO tblPhone(Фамилия, Телефон) " & _
                      "VALUES('Иванова','121-11-11')"
    Set cmd.ActiveConnection = cn
    cmd.Execute
    Set cnd = Nothing
    cn.Close
    Set cn = Nothing
End Sub
```

ГЛАВА 16



Интеграция приложений

В данной главе описывается мощная технология Automation, которой обладает VBA, и которая существенно упрощает процесс и сокращает время разработки проектов. Технология Automation предоставляет разработчику доступ к объектам и методам других приложений, максимально приспособленных для решения специализированных задач. Теперь разработчику нет нужды каждый раз "изобретать велосипед", а достаточно просто воспользоваться технологией Automation. В этой главе демонстрируется, как можно интегрировать работу MS Excel с MS Word, MS Access и MS PowerPoint. В моей книге "Excel, VBA и Internet в экономике и финансах", опубликованной издательством "БХВ-Петербург", приведены также примеры интеграции MS Excel с MS Outlook, Visual Basic 6.0 и построением компонентов в Visual Basic 6.0 для последующего их использования в офисных приложениях.

Технология ActiveX

Каждая из программ MS Office является очень гибким продуктом, предоставляющим пользователю большие возможности. Используя эти возможности, можно создать более эффективные и технологичные приложения. Технология ActiveX как раз и реализует взаимодействие различных приложений MS Office друг с другом, а также с любыми программами, поддерживающими эту технологию, например, с Visual Basic при помощи компонентной объектной модели (Component Object Model, COM). Ранее эта технология называлась OLE (Object Linking and Embedding, связывание и внедрение объектов). В настоящее время OLE является частью технологии ActiveX.

Главным предназначением ActiveX является обеспечение простого и эффективного взаимодействия различных продуктов. Например, MS Word или MS Excel могут предоставить свои объекты друг другу. Это позволяет создавать документы, состоящие из элементов, созданных в различных приложениях.

Технология ActiveX предоставляет разработчику два мощных средства для создания приложений:

- OLE — связывание и внедрение объектов разных приложений. Это средство позволяет непосредственно в документе редактировать объекты, созданные другими приложениями;
- Automation, которое позволяет программно управлять как внедренными объектами, так и объектами других приложений. Таким образом, значительно сокращается время разработки, т. к. нет нужды заново реализовывать те возможности, которые не включены в качестве встроенных средств, но которые имеются у других специализированных приложений.

Примечание

Технология ActiveX подразумевает, что интегрируемые приложения, объекты которых используются в документе, установлены и зарегистрированы в системном реестре. Регистрация приложений, как правило, происходит автоматически при их установке.

Связывание и внедрение объектов

Технология ActiveX позволяет внедрять объект, созданный в одном приложении в документ, созданный другим приложением. Например, можно внедрить рабочий лист или диаграмму MS Excel в документ MS Word. Конечно, можно сделать и наоборот, в рабочий лист MS Excel внедрить документ MS Word. Объектом называется произвольный элемент, созданный в каком-либо одном приложении (*приложении-источнике* или *сервере*), который можно поместить в документ другого приложения (*приложения-приемника* или *контейнера*), причем сделать это так, что вместе со вставленными данными будет храниться информация о приложении, создавшем этот объект. Впоследствии это дает возможность редактировать объект средствами создавшего его приложения. При *внедрении* все данные копируются в документ-контейнер. При *связывании* в контейнер записывается только информация о приложении-источнике, из которого взят вставляемый объект.

В документ можно внедрять как еще не существующий (новый) объект, так и уже существующий.

Внедрение нового объекта

При внедрении нового объекта с помощью OLE, он создается непосредственно в документе. Например, объект Microsoft WordArt 2.0 — для создания фигурного текста с помощью приложения MS WordArt, объект Microsoft Map — для создания карты.

Внедрим, например, в рабочий лист MS Excel объект (рисунок) из MS Clip Gallery. Первоначально, конечно, надо определиться с местом в документе, куда будет внедряться объект. В MS Excel, например, нужно выбрать ячейку, которая будет служить верхним левым углом внедряемого объекта. После этого выберите команду **Вставка | Объект**. На экране отобразится диалоговое окно **Вставка объекта**. На вкладке **Новый** этого диалогового окна в списке **Тип объекта** выберите объект MS Clip Gallery. Убедитесь, что снят флажок **В виде значка**. Нажмите кнопку **OK**. На экране отобразится внедренный объект. Внедрять можно как сам объект, так и его значок. Тогда для просмотра объекта надо будет просто дважды щелкнуть на соответствующем значке. Значки применяются при внедрении громоздких объектов, объектов мультимедиа и т. д. Для внедрения не объекта, а значка объекта надо на вкладке **Новый** диалогового окна **Вставка объекта** установить флажок **В виде значка**. Значок можно изменить, нажав на кнопку **Сменить значок**, что приведет к отображению диалогового окна **Смена значка**. Внедренный объект можно перемещать по документу, а его размеры можно изменять стандартным для приложений Windows способом, используя маркеры перемещения и изменения размера, соответственно.

Внедрение существующего объекта

Таким же способом, как внедряется новый объект, в документ можно внедрять объект, представляющий собой файл, хранящийся на диске, т. е. уже существующий объект. Для этого надо воспользоваться вкладкой **Из файла** диалогового окна **Вставка объекта**. Например, внедрим файл Word в рабочий лист. Выберите команду **Вставка | Объект**. На экране отобразится диалоговое окно **Вставка объекта**. На вкладке **Из файла** этого диалогового окна в поле **Имя файла**, используя кнопку **Обзор**, введите имя внедряемого файла. Убедитесь, что снят флажок **В виде значка** и установлен флажок **Связь с файлом**. Нажмите кнопку **OK**. На рабочем листе отобразится внедренный объект.

Установка флажка **Связь с файлом** приведет к внедрению объекта со связью с файлом. Теперь любые изменения в файле будут автоматически приводить и к изменениям в документе.

Технология Automation

Automation — это ключевая технология, используемая в большинстве средств разработки корпорации Microsoft. Она дает возможность программно управлять объектами из других приложений, т. е. позволяет разработчикам использовать объекты (и все, что с ними связано) других приложений в качестве компонентов своих собственных приложений. Одним из ключевых свойств технологии Automation является ее независимость от языка программирования.

Приложения поддерживают технологию Automation одним из двух способов:

- в качестве приложения-источника (или *объекта*, или *сервера Automation*, или *объекта ActiveX*) — объекты такого приложения используются другим приложением или средствами программирования через интерфейс программирования;
- в качестве приложения-приемника (или *контроллера*, или *клиента Automation*) — это приложение управляет объектами приложения-источника.

Примечание

Некоторые приложения могут быть только приложениями-источниками либо только приложениями-приемниками, но есть и такие, которые могут выступать и в той и в другой роли (например, MS Excel, MS Access).

Для программного управления объектом Automation надо:

1. Создать переменную, представляющую собой объект.
2. Использовать эту переменную для доступа к объектам, находящимся в приложениях-источниках. Объекты приложения источника образуют библиотеку *объектов-серверов*.
3. По завершении работы с объектом присвоить переменной значение `Nothing`, освобождая переменную.

Программные идентификаторы приложений-серверов Automation

В табл. 16.1 приведены названия приложений, типов и классов объектов Automation, а также их программные идентификаторы, которые иногда называются ProgID, используемые при программировании объектов Automation. Обратите внимание, что если применяется программный идентификатор без указания версии, то объект создается на основе наиболее современной установленной версии программы. В общем случае идентификатор ProgID имеет следующий синтаксис:

Appname.ObjectType

- *Appname* — имя приложения сервера;
- *ObjectType* — тип или класс объектов.

Таблица 16.1. *Имена серверов Automation*

Приложение	Тип объекта	Идентификатор
Excel	`Application`	`Excel.Application`
	`Workbook`	`Excel.Sheet`
	`Workbook`	`Excel.Chart`

Таблица 16.1 *(окончание)*

Приложение	Тип объекта	Идентификатор
Access	Application	Access.Application
	CurrentData	Access.CodeData, Access.CurrentData
	CurrentProject	Access.CodeProject, Access.CurrentProject
Word	Application	Word.Application
	Document	Word.Document, Word.Template
PowerPoint	Application	PowerPoint.Application
Outlook	Application	Outlook.Application

Функции доступа к объектам Automation

Для доступа к объектам Automation приложения-сервера используются две функции: `CreateObject` и `GetObject`.

Функция `CreateObject` возвращает и создает ссылку на объект ActiveX.

```
CreateObject(Class,[Servername])
```

- *`Class`* — имя объекта Automation;
- *`Servername`* — параметр используется только при создании объекта Automation в сети и устанавливает имя сервера, где будет создан объект Automation.

Функция `GetObject` возвращает и создает ссылку на объект ActiveX, сохраненный в файле.

```
GetObject([Pathname] [, Class])
```

- *`Pathname`* — полное имя файла; если параметр опущен, то необходимо указать значение параметра *`Class`*;
- *`Class`* — имя объекта Automation.

Функция `GetObject` подобна функции `CreateObject`. Но есть и некоторое различие между ними. Функцию `GetObject` можно использовать для доступа к существующим документам, хранящимся в файлах, а также и для доступа к объекту `Application` любого уже запущенного приложения MS Office. Для этого надо вызвать функцию `GetObject` без первого параметра. Этот способ доступа к объекту `Application` любого уже запущенного приложения

MS Office применяется, когда нет необходимости в запуске еще одного экземпляра приложения, что происходит при работе функции `CreateObject`.

Позднее и ранее связывание

Позднее связывание (late binding) происходит, когда тип для переменной, которая будет представлять собой объект Automation, указывается как `Object`.

Таким образом, при позднем связывании переменная, задающая объект Automation, имеет тип `Object`. Тип `Object` позволяет создавать объекты любой природы. В этом смысле он подобен типу `Variant`. Такая чрезмерная общность определения переменной понижает производительность приложения. Для достижения наилучшей производительности приложения необходимо определить конкретный тип для переменной, которая будет представлять собой объект Automation. Например, если используется Excel, то надо установить тип переменной `Excel.Application`.

Второй подход называется *ранним связыванием* (early binding) и происходит на этапе компиляции. При раннем связывании, чтобы определить переменную определенного класса, перед написанием кода, необходимо сослаться на библиотеку объектов серверов Automation. Для этого в редакторе Visual Basic выберите команду **Tools | References**. В появившемся диалоговом окне **References** в списке **Available References** установите флажок подсоединяемой библиотеки объектов, например **Microsoft Word 10.0 Object Library**.

Выполнив предварительные действия, можно перейти к созданию нового экземпляра объекта Automation при помощи ключевого слова `New`. Например:

```
Dim objWorkbook As New Word.Application
```

Совет

Если вы хотите, чтобы при написании кода после набора точки на экране отображался список со свойствами и методами объекта Automation, то используйте раннее связывание. Кроме того, раннее связывание позволяет использовать константы подсоединяемой объектной модели без их объявления.

Предупреждение

К недостаткам раннего связывания можно отнести привязанность создаваемого вами приложения к конкретной версии библиотеки объектов. Выход в свет новой версии продукта, объектную модель которого вы используете посредством раннего связывания, приведет к необходимости повторного распространения и вашего обновленного продукта. Проекты, созданные с помощью позднего связывания, не зависят от новых версий используемых объектных моделей.

Microsoft Excel и Microsoft Word

MS Word является одним из наиболее распространенных средств по подготовке документов и отчетов, когда-либо созданных для Windows, и любой программист, конечно, захочет использовать его возможности в своих проектах. В данном разделе на примерах показано, как, используя объектную модель MS Word, можно создавать отчетную документацию из MS Excel.

Создание нового документа Microsoft Word функцией *CreateObject*

На простом примере покажем, как открывается документ MS Word (в данном случае файл Пример.doc, который следует расположить в рабочем каталоге проекта). Первоначально программа проверяет наличие файла Пример.doc в рабочем каталоге приложения. Если его там нет, то программа информирует об этом пользователя и прерывает свое выполнение. Если же файл есть, то программа с помощью функции `CreateObject` запускает еще один экземпляр MS Word, даже если MS Word уже был запущен, и в нем открывается файл. По открытии файла на экране отображается диалоговое окно с сообщением **Закрыть документ?** Нажатие кнопки **OK** на этом окне приведет как к закрытию окна, так и созданного экземпляра Word с загруженным файлом. Нажатие кнопки **Нет** оставит документ открытым. Для реализации данного демонстрационного приложения создайте форму, в которой расположите кнопку, установите значение ее свойства `Name` равным `cmdCreateNew`, а в модуле формы наберите следующий код (листинг 16.1).

Листинг 16.1. Открытие документа Word при помощи функции `CreateObject`

```
Private Sub cmdCreateNew_Click()
   Dim objWord As Object
   Dim File As String
   File = ThisWorkbook.Path & "\Пример.doc"
   If Dir(File) = Empty Then
      MsgBox "Документ не найден", vbInformation, "Word"
      Exit Sub
   End If
   Set objWord = CreateObject("Word.Application")
   With objWord
      .Visible = True
      .Documents.Open Filename:=File
   End With
   Select Case MsgBox("Закрыть документ", vbQuestion + vbYesNo, "Word")
      Case vbYes
```

```
            objWord.Quit
            Set objWord = Nothing
        Case vbNo
            Set objWord = Nothing
    End Select
End Sub
```

Совет

При работе с MS Word с помощью технологии Automation необходимо не забыть закрыть в коде приложение, т. к. в противном случае по умолчанию оно остается открытым. Закрытие MS Word производится методом `Quit`. После того как MS Word закрыт, можно уничтожить созданный объект `objWord`.

Открытие документа Microsoft Word функцией *GetObject*

В данном разделе мы не только покажем, как открывается существующий документ MS Word, но и объясним, как проверить, запущен ли уже MS Word. Для этого применим функцию `GetObject`. Если Word запущен, то для открытия файла можно воспользоваться функцией `GetObject`. Если же нет, то использование функции `GetObject` приведет к генерированию ошибки `429`. В таком случае надо применить функцию `CreateObject`. Кроме того, в нашей программе предусмотрим печать открытого документа, при этом программа первоначально спрашивает пользователя, надо ли печатать документ. Документ печатается, если только пользователь подтвердит необходимость печати. По завершении печати документ закрывается, а также закрывается MS Word, если он был открыт программой для печати документа. Следует обратить внимание на то, что перед выполнением программы (листинг 16.2) надо установить ссылку на библиотеку объектов Microsoft Word 10.0 Object Library в окне **References**, отображаемом на экране выбором в редакторе Visual Basic команды **Tools | References**.

Листинг 16.2. Открытие документа MS Word при помощи функции `GetObject` и вывод его на печать

```
Private objWord As Word.Application
Private Flag As Boolean

Private Sub cmdCreateNew_Click()
   Dim Button As Integer, Test As Integer, File As String
   File = ThisWorkbook.Path & "\Пример.doc"
   Flag = False
   On Error GoTo ErrorHandler
   Set objWord = GetObject(, "Word.Application")
```

```
   objWord.Application.DisplayAlerts = wdAlertsNone
   With objWord
      .Visible = True
      .Documents.Open Filename:=File
   End With
   Button = vbYesNo + vbQuestion
   If MsgBox("Печатать документ?", Button, File) = vbYes Then
      objWord.ActiveDocument.PrintOut
      Do While objWord.BackgroundPrintingStatus <> 0
         DoEvents
      Loop
   End If
   Destructor
   Exit Sub

ErrorHandler:
   Select Case Err.Number
      Case 429
         Flag = True
         Set objWord = CreateObject("Word.Application")
         Resume Next
      Case Else
         MsgBox Err.Description & " " & Err.Number, vbInformation
         Exit Sub
    End Select
    Destructor
End Sub

Private Sub Destructor()
   objWord.ActiveDocument.Close
   If Flag Then objWord.Quit
   Set objWord = Nothing
End Sub
```

Примечание

Обратите внимание, что данный пример демонстрирует не только то, как выводится на печать документ, но и то, как проверяется, открыт ли уже экземпляр приложения или нет.

Создание отчета из MS Excel в MS Word

Продемонстрируем, как можно построить отчетный документ MS Word на основе данных, полученных при расчетах в MS Excel (рис. 16.1). В качестве примера создадим приложение, моделирующее бросание кости. Игрок в поле вводит число попыток и нажимает кнопку **Игра**. В список выводится

результат игры. При нажатии же кнопки **Отчет** создается новый документ MS Word, в котором на основе данных из списка конструируется таблица результатов игры.



Рис. 16.1. Создание отчета из MS Excel в MS Word

Итак, создайте форму. Расположите на ней две кнопки и список. При помощи окна **Properties** установите элементам управления значения свойств, как показано в табл. 16.2.

Таблица 16.2. *Значения свойств, установленные в окне* ***Properties***

Элемент управления	Свойство	Значение
Форма	Caption	Игра
Кнопка	Name	cmdGame
	Caption	Игра

Таблица 16.2 *(окончание)*

Элемент управления	Свойство	Значение
Кнопка	Name	cmdReport
	Caption	Отчет
Список	Name	lstRes

Для корректного функционирования программы (листинг 16.3) установите ссылку на библиотеку объектов Microsoft Word 10.0 Object Library в окне **References**, отображаемом на экране выбором в редакторе Visual Basic команды **Tools | References**.

Листинг 16.3. Создание отчета из MS Excel в MS Word

```
Private Sub UserForm_Initialize()
   lstRes.ColumnCount = 2
End Sub

Private Sub cmdGame_Click()
   PlayGame
End Sub

Private Sub cmdReport_Click()
   ReportBuilder
End Sub

Private Sub PlayGame()
   Dim n As Integer, i As Integer
   n = CInt(txtNum.Text)
   Randomize
   lstRes.Clear
   For i = 0 To n - 1
      lstRes.AddItem i + 1
      lstRes.List(i, 1) = CInt(5 * rnd()) + 1
   Next
End Sub

Private Sub ReportBuilder()
   Dim n As Integer
   n = lstRes.ListCount
   If n = 0 Then Exit Sub
```

```
Dim objWord As New Word.Application
Dim objDoc As Word.Document
With objWord
   .WindowState = wdWindowStateMaximize
   .Visible = True
   .Documents.Add
   Set objDoc = .ActiveDocument
End With
With objWord.Selection
   .Font.Bold = wdToggle
   .ParagraphFormat.Alignment = wdAlignParagraphCenter
   .TypeText Text:="Результат"
   .TypeParagraph
   .TypeText Text:="игры в кость"
   .InsertParagraphAfter
   .MoveRight
End With
With objDoc.Paragraphs.Item(1).Range
   .ParagraphFormat.Alignment = wdAlignParagraphCenter
   With .Font
      .Size = 18
      .Bold = True
   End With
End With
With objDoc.Paragraphs.Item(2).Range
   .ParagraphFormat.Alignment = wdAlignParagraphCenter
   With .Font
      .Size = 14
      .Bold = True
   End With
End With

With objDoc
   .Tables.Add Range:=objWord.Selection.Range, _
               NumRows:=n + 1, NumColumns:=3
   .Tables.Item(1).AutoFormat Format:=wdTableFormatColorful2
End With
With objWord.Selection
   .TypeText "Номер попытки"
   .MoveRight Unit:=wdCell
   .TypeText "Очки"
   .MoveRight Unit:=wdCell
   .TypeText "Комментарии"
```

```
      .MoveRight Unit:=wdCell
   End With

   Dim i As Integer
   Dim r As Integer
   Dim msg As String

   For i = 0 To n - 1
      With objWord.Selection
         .TypeText Text:=i
         .MoveRight Unit:=wdCell
         r = lstRes.List(i, 1)
         .TypeText Text:=r
         .MoveRight Unit:=wdCell
         If r = 6 Then
            msg = "Выигрыш!"
         Else
            msg = "Не везет..."
         End If
         .TypeText Text:=msg
         .MoveRight Unit:=wdCell
      End With
   Next
   objWord.Selection.Cells.Delete ShiftCells:=wdDeleteCellsEntireRow
   Set objWord = Nothing
End Sub
```

MS Excel и MS Access

MS Access является мощным средством по созданию баз данных и работе с информацией, в них хранящейся. В частности, этот программный продукт весьма эффективен при создании удобного визуального интерфейса по работе с базой данных и составлению отчетов.

Создание объектов MS Access

Продемонстрируем, как технология Automation позволяет открыть таблицу базы данных. В качестве примера взята учебная база данных Борей.mdb, поставляемая вместе с MS Office. Файл Борей.mdb следует расположить в корневом каталоге диска C:. Кроме того, установите ссылку на библиотеку объектов Microsoft Access 10.0 Object Library и в модуле рабочего листа наберите следующий код (листинг 16.4). Открытие таблицы **Сотрудники** базы

данных Борей.mdb будет происходить при щелчке правой кнопкой мыши на ячейке **A1**.

Листинг 16.4. Создание объектов Access. Модуль рабочего листа

```
Private Sub Worksheet_BeforeRightClick(ByVal Target As Range, _
                                       Cancel As Boolean)
   If Target.Address = "$A$1" Then
      calcel = True
      Dim objAccess As Access.Application
      Set objAccess = CreateObject("Access.Application")
      On Error GoTo ErrorHandler
      With objAccess
         .OpenCurrentDatabase "c:\Борей.mdb"
         .DoCmd.OpenTable "Сотрудники"
         .Visible = True
      End With
      Set objAccess = Nothing
      Exit Sub
   End If

ErrorHandler:
   On Error Resume Next
   Set objAccess = Nothing
End Sub
```

Печать или предварительный просмотр отчета

В данном разделе демонстрируется на примере не только как технология Automation позволяет открыть базу данных, но и как напечатать или предварительно просмотреть отчет, созданный в базе данных.

Создайте форму, на которой расположите две кнопки. Установите значения свойств `Caption` и `Name` этих кнопок равными `Print`, `cmdPrint`, `Preview`, `cmdPreview`. Кроме того, надо установить ссылку на библиотеку объектов Microsoft Access 10.0 Object Library, а в модуле формы набрать следующий код (листинг 16.5).

Предварительный просмотр отчета **Итоги продаж по объему** базы данных Борей.mdb программа производит при нажатии кнопки **Preview**, а его печать — при нажатии кнопки **Print**.

Листинг 16.5. Печать или предварительный просмотр отчета

```
Private Sub UserForm_Initialize()
   strReport = "Итоги продаж по объему"
```

```
   Set objAccess = CreateObject("Access.Application")
   On Error GoTo ErrorHandler
   With objAccess
      .Visible = True
      .OpenCurrentDatabase ThisWorkbook.Path & "\Борей.mdb"
   End With
Exit Sub
ErrorHandler:
   On Error Resume Next
   Set objAccess = Nothing
End Sub

Private Sub cmdPrint_Click()
   DoIt strReport, cPrint
End Sub

Private Sub cmdPreview_Click()
   DoIt strReport, cPreview
End Sub

Private Sub DoIt(srtName As String, Flag As PrintPreview)
   Select Case Flag
      Case cPrint
         objAccess.DoCmd.OpenReport srtName, acNormal
      Case cPreview
         objAccess.DoCmd.OpenReport srtName, acPreview
   End Select
End Sub
```

Экспорт данных из MS Access в MS Excel

Рассмотрим пример экспорта данных из MS Access в MS Excel. Выбранная запись будет экспортироваться на рабочий лист. Для осуществления проекта в MS Access создайте базу данных, состоящую из одной таблицы **Путевки**, содержащей четыре поля: **Номер**, **Фамилия**, **Страна** и **Оплата**.

Создайте форму с указанием таблицы **Путевки** как источника данных. Расположите на ней четыре надписи и поля, связав их соответственно с полями **Номер**, **Фамилия**, **Страна** и **Оплата** (рис. 16.2). Кроме того, создайте кнопку и установите у нее значения свойств `Имя` и `Подпись` равными `cmdExcel` и `Ввод в рабочий лист`. Для кнопки создайте обработчик события `Нажатие кнопки` и наберите код, представленный в листинге 16.6. Проект готов.

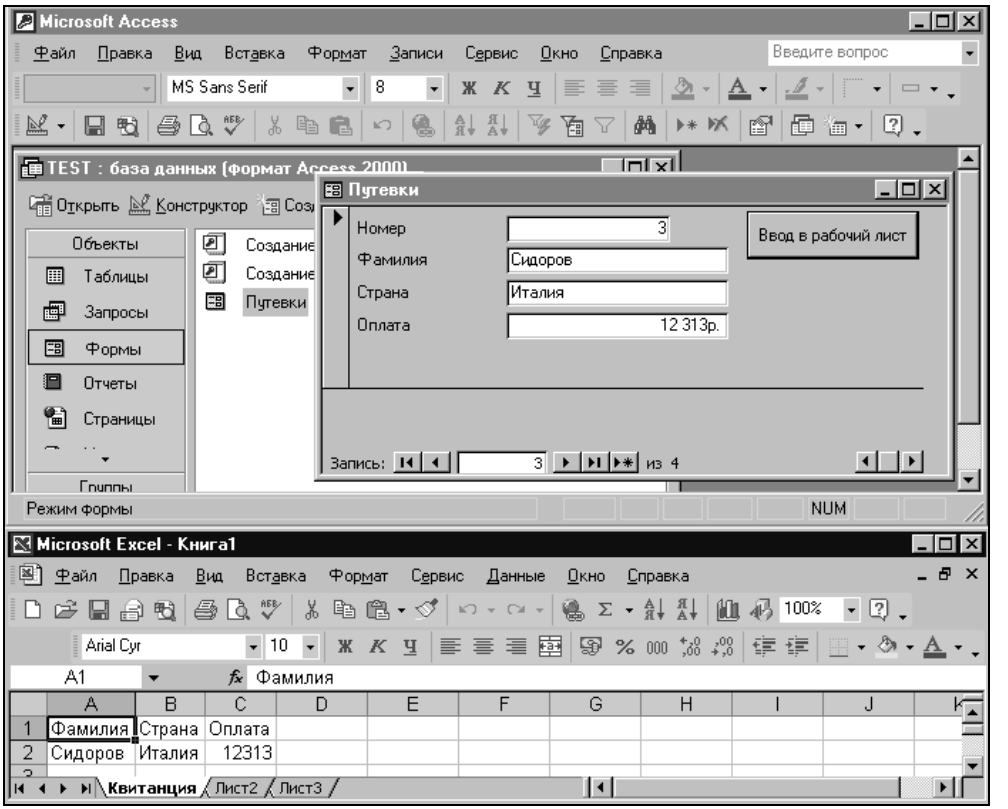


Рис. 16.2. Экспорт данных из MS Access в MS Excel

Листинг 16.6, *а*. Экспорт данных из MS Access в MS Excel. Модуль формы

```
Option Compare Database

Private Sub cmdExcel_Click()
    ExportExcel
End Sub
```

Листинг 16.6, *б*. Экспорт данных из MS Access в MS Excel. Стандартный модуль

```
Public Type Tourist
    Name As String
    Country As String
    Payment As String
End Type
```

```
Public Sub ExportExcel()
   Dim newTourist As Tourist
   Dim objExcel As Object
   With newTourist
      .Name = Forms(0)("Фамилия")
      .Country = Forms(0)("Страна")
      .Payment = Forms(0)("Оплата")
   End With
   Set objExcel = CreateObject("Excel.Application")
   With objExcel
      .Workbooks.Add
      .ActiveSheet.Name = "Квитанция"
      .Visible = True
      With .Worksheets("Квитанция")
         .Range("A1").Value = "Фамилия"
         .Range("A2").Value = newTourist.Name
         .Range("B1").Value = "Страна"
         .Range("B2").Value = newTourist.Country
         .Range("C1").Value = "Оплата"
         .Range("C2").Value = newTourist.Payment
         .Columns("A:C").AutoFit
      End With
   End With
End Sub
```

Создание презентации в MS PowerPoint

MS PowerPoint является удобным средством по созданию презентаций. Использование же технологии ActiveX позволяет автоматизировать процесс конструирования слайдов для таких презентаций. Например, следующая процедура (листинг 16.7) из стандартного модуля рабочей книги на основе таблицы, расположенной в диапазоне **A1:B8**, и диаграммы создает презентацию, состоящую из двух слайдов, а затем начинает показ созданного слайда (рис. 16.3).

Листинг 16.7. Создание презентации в MS PowerPoint

```
Sub PowerPointPresentation()
   Dim pp As Object, prs As Object
   Set pp = CreateObject("PowerPoint.Application")
   Worksheets(1).Range("A1:B8").Copy
   Set prs = pp.Presentations.Add
   Dim ppLayoutBlank As Integer
   ppLayoutBlank = 12
```

```
    prs.Slides.Add 1, ppLayoutBlank
    pp.Visible = True
    pp.ActiveWindow.View.Paste

    Worksheets(1).ChartObjects(1).Copy
    Dim ppLayoutText As Integer
    pp.ActiveWindow.View.GotoSlide _
       Index:=pp.ActivePresentation.Slides.Add( _
                Index:=2, Layout:=ppLayoutBlank).SlideIndex
    pp.ActiveWindow.View.Paste

    pp.ActivePresentation.SlideShowSettings.Run
    Set pp = Nothing
End Sub
```

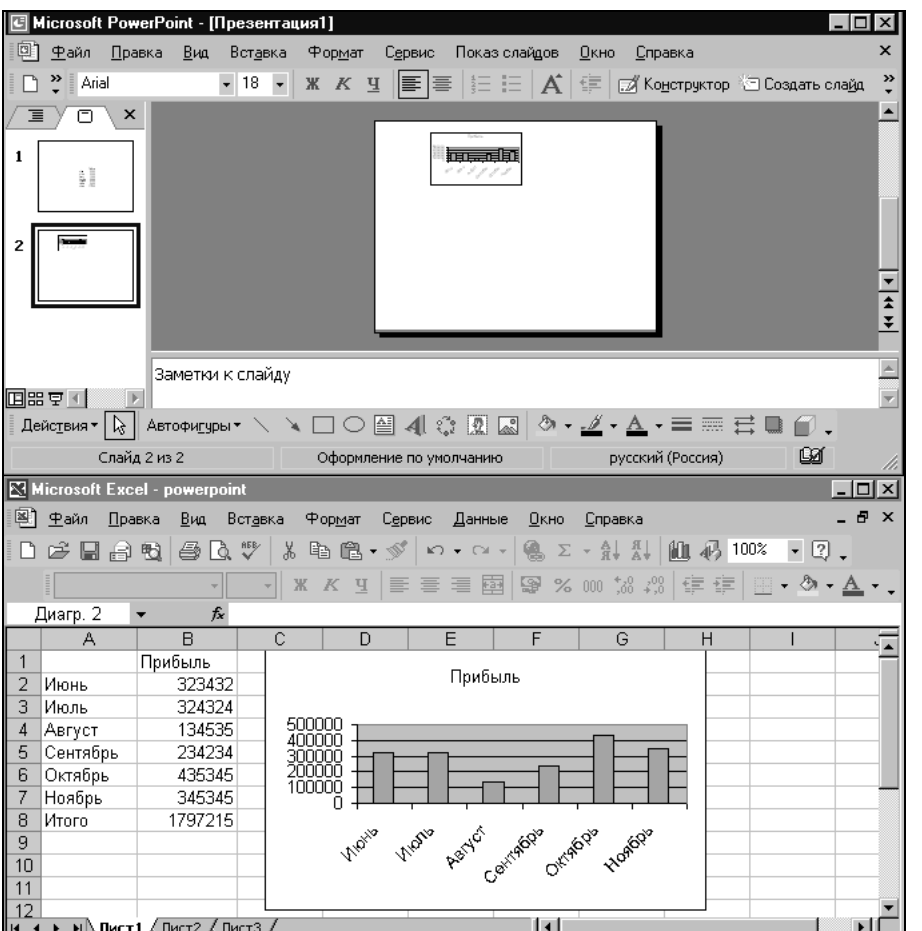
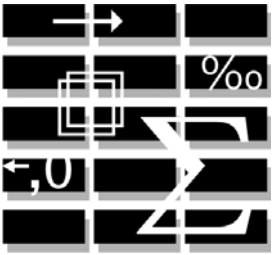


Рис. 16.3. Создание презентации в MS PowerPoint из MS Excel

ГЛАВА 17



MS Excel и Internet

Гиперссылки

Гиперссылки (hyperlinks) — это прекрасное дополнение к MS Excel. Они позволяют совершать "прыжки" к другим документам и объектам Windows. Например, можно сделать гиперссылку на другой рабочий лист MS Excel, офисный документ другого типа или Web-страницу. Если кто-то читает рабочий лист, то гиперссылки позволяют направлять его перемещение по документу или документам. Например, в ячейке документа можно создать гиперссылку, которая отсылает к документу MS Word. Если пользователю нужно больше информации о документе, то можно щелкнуть на ссылке и автоматически переместиться к необходимому документу или узлу Internet.

Гиперссылки выглядят как ячейки, содержащие подчеркнутый текст. Когда указатель мыши перемещается над ячейками с гиперссылками, он принимает форму руки с указывающим пальцем. При щелчке на гиперссылке открывается связанный документ. Если у вас есть доступ к Internet, гиперссылки на Web-узлы откроют Internet Explorer и доступ к соответствующему Web-узлу. Например, информация о приложениях Microsoft Office содержится на узле **http://www.microsort.com**, а о книгах издательства "БХВ-Петербург" — **http://www.bhv.ru**.

Если конкретный файл в URL (Uniform Resource Locator, унифицированный локатор ресурсов) не указан, то открывается страница по умолчанию для данного Web-узла.

При переходе между страницами с помощью гиперссылки сохраняется хронология просмотра всех страниц.

Добавление гиперссылок на документы MS Office

Часто в работе приходится использовать документы, созданные в различных приложениях MS Office. Например, добавляя гиперссылки на рабочие листы, можно перемещаться между этими документами одним щелчком.

Чтобы на рабочем листе MS Excel создать гиперссылку на другой документ MS Office (в том числе и на текущую рабочую книгу, но на другой диапазон ячеек или другой рабочий лист):

1. Выберите диапазон или фигуру, с которой гиперссылка будет связана.
2. Выберите команду **Вставка | Гиперссылка**.

   На экране отобразится окно **Добавление гиперссылки** (рис. 17.1). Переключатели **файлом**, **веб-страницей**, **местом в документе**, **новым документом**, **электронной почтой** группы **Связать с** указывают документ, на который будет дана ссылка в гиперссылке. В нашем случае выберите переключатель **файлом**, **веб-страницей**. Поле **Текст** задает текст гиперссылки. В поле **Папка** выбирается папка, в которой лежит искомый документ. В списках **текущая папка**, **просмотренные страницы**, **последние файлы** можно выбрать искомый файл. В списке **Адрес** можно указать URL документа, а кнопка **Закладка** позволит выбрать конкретное место, в которое будет производиться переход по гиперссылке. После установки необходимых параметров нажмите кнопку **OK**.

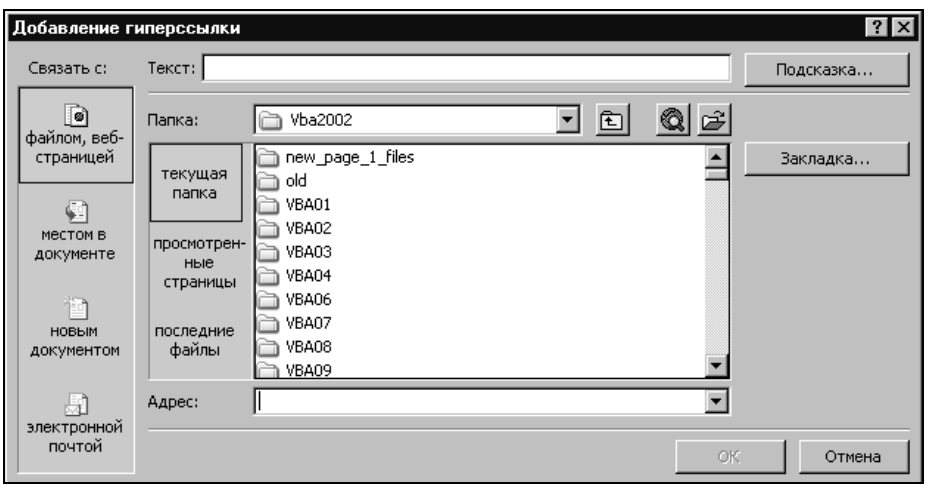


Рис. 17.1. Окно **Добавление гиперссылки**

Отредактировать гиперссылку можно, щелкнув на ней правой кнопкой мыши. На экране отобразится контекстное меню, которое предлагает три команды по работе с гиперссылками:

- **Изменить гиперссылку** — открывает окно **Изменение гиперссылки**. В нем можно изменить путь к связанному файлу и его имя, а также добавить пояснительное примечание к гиперссылке;
- **Открыть гиперссылку** — открывает связанный документ, как это делается при щелчке на гиперссылке;
- **Удалить гиперссылку** — удаляет гиперссылку.

Задание гиперссылки формулой рабочего листа

Гиперссылку можно создавать формулой рабочего листа ГИПЕРССЫЛКА.

ГИПЕРССЫЛКА(*адрес*;*имя*)

- *адрес* — это путь и имя файла для открываемого документа. Адрес может ссылаться на место в документе — например, на определенную ячейку или именованный интервал на листе книги MS Excel или на закладку в документе MS Word. Путь может представлять собой путь к файлу, записанному на жестком диске, может также быть адресом в формате UNC сервера (в MS Excel для Windows) или адресом URL в Internet или внутренней сети;
- *имя* — текст перехода или численное значение, отображаемое в ячейке. Имя отображается синим цветом и с линией подчеркивания. Если аргумент опущен, ячейка в качестве текста перехода отображает аргумент *адрес*.

Примечание

В качестве значений параметров функции ГИПЕРССЫЛКА могут быть либо текстовые выражения, либо ссылка на ячейку.

Так, если в книге Department.xls из ячейки, например **B3**, необходимо перейти в ячейку **A20** рабочего листа **Отчет** (кстати, ячейка **B3** может сама располагаться на рабочем листе **Отчет**, т. е. допустимо движение как внутри рабочего листа, так и между листами), то в ячейку **B3** надо ввести гиперссылку

```
=ГИПЕРССЫЛКА("[Department.xls]Отчет!A20";"Перейти в отчет")
```

либо формулу

```
=ГИПЕРССЫЛКА(A1;B1)
```

где в ячейку **A1** введено `[Department.xls]Отчет!A20`, а в ячейку **B1** — `Перейти в отчет`.

Первая из следующих двух гиперссылок открывает рабочую книгу Sales.xls, находящуюся в корневом каталоге диска D:, а вторая — открывает эту книгу и активизирует рабочий лист **Май**:

```
=ГИПЕРССЫЛКА("D:\Sales.xls";"Продажи")
=ГИПЕРССЫЛКА("[D:\Sales.xls]Май!A1";"Продажи")
```

Гиперссылку на Web-страницу корпорации Microsoft можно организовать, например, следующим образом:

```
=ГИПЕРССЫЛКА("http://www.microsort.com","Microsort")
```

Условная гиперссылка

Чтобы гиперссылка включалась или выключалась в зависимости от какого-либо условия, используйте ее в композиции с функцией `ЕСЛИ`. Например:

```
=ЕСЛИ(C1="Май";ГИПЕРССЫЛКА("[D:\Sales.xls]Май!A1";"Продажи");"")
```

Автоматическое изменение гиперссылки в зависимости от значения в какой-либо ячейки легко реализуется при помощи функции `ЕСЛИ` или `ВЫБОР`. Например:

```
=ВЫБОР(B1; ГИПЕРССЫЛКА("D:\May.xls";"Май");
↳ГИПЕРССЫЛКА("D:\June.xls";"Июнь");
↳ГИПЕРССЫЛКА("D:\July.xls";"Июль"))
```

Объект *Hyperlink* и семейство *Hyperlinks*

Гиперссылки в VBA представлены объектом `Hyperlink`, который является элементом семейства `Hyperlinks`, состоящего из всех гиперссылок рабочего листа или диапазона. Это семейство имеет два метода `Add` (добавить новую гиперссылку в семейство) и `Delete` (удалить все гиперссылки из семейства).

```
Add(Anchor, Address, SubAddress, ScreenTip, TextToDisplay)
```

- *Anchor* — задает местоположение гиперссылки. Может быть либо объектом `Range`, либо объектом `Shape`;
- *Address* — адрес гиперссылки;
- *SubAddress* — область в документе (например, диапазон ячеек или закладка), на которую происходит переход;
- *ScreenTip* — текст всплывающей подсказки;
- *TextToDisplay* — текст, отображаемый на гиперссылке.

Пример создания новой гиперссылки приведен в листинге 17.1.

Листинг 17.1. Создание новой гиперссылки

```
Private Sub Workbook_Open()
   With Worksheets(1)
    .Hyperlinks.Add Anchor:=.Range("A1"), _
                    Address:="D:\Sales.xls", _
                    ScreenTip:="ООО Рога и копыта", _
                    TextToDisplay:="Продажи"
   End With
End Sub
```

В табл. 17.1 перечислим свойства, а табл. 17.2 — методы объекта `Hyperlink`.

Таблица 17.1. *Свойства объекта* `Hyperlink`

Свойство	Описание
`Address`	Адрес гиперссылки
`EmailSubject`	Передаваемая строка текста
`Range`	Возвращает объект `Range`, которому назначена гиперссылка
`ScreenTip`	Текст всплывающей подсказки
`Shape`	Возвращает объект `Shape`, которому назначена гиперссылка
`SubAddress`	Область в документе, на которую происходит переход
`TextToDisplay`	Текст, отображаемый на гиперссылке
`Type`	Возвращает объект, к которому подсоединена гиперссылка. Допустимые значения: `msoHyperlinkInlineShape`, `msoHyperlinkRange`, `msoHyperlinkShape`

Таблица 17.2. *Методы объекта* `Hyperlink`

Метод	Описание
`AddToFavorites`	Добавить в список избранных ссылок
`CreateNewDocument`	Создать новый документ, связанный с указанной гиперссылкой
`Delete`	Удалить гиперссылку
`Follow`	Перейти по специфицированной гиперссылке

Переход по гиперссылке из списка

Гиперссылки располагаются на рабочем листе. А можно ли переходить по гиперссылке, указанной в списке? Да, конечно. Для этого достаточно на рабочем листе создать настоящие гиперссылки. Затем спрятать их, например, за списком. Заполнить список и создать соответствующие объекты `Hyperlink`. А далее применить метод `Follow` для перехода по указанной гиперссылке. Например, в следующем коде (листинг 17.2) при двойном щелчке на элементе списка, расположенном на рабочем листе, производится переход по гиперссылке на книгу May.xls или June.xls.

Листинг 17.2, *а*. Переход по гиперссылке из списка. Модуль `ЭтаКнига`

```
Private Sub Workbook_Open()
   Worksheets(1).ListBox1.ColumnCount = 2
   Worksheets(1).ListBox1.ColumnWidths = "100;0"
   Worksheets(1).ListBox1.Clear
   Dim lst(1, 1) As String
   lst(0, 0) = "May report": lst(0, 1) = "may.xls"
   lst(1, 0) = "June report": lst(1, 1) = "june.xls"
   Worksheets(1).ListBox1.List = lst
   Dim i As Integer
   Dim r As Hyperlink
   For Each r In Worksheets(1).Hyperlinks
      r.Delete
   Next
   For i = 0 To 1
    Worksheets(1).Hyperlinks.Add Anchor:=Worksheets(1).Cells(i + 1, 1), _
                                 Address:=lst(i, 1), _
                                 TextToDisplay:=lst(i, 0)
   Next
End Sub
```

Листинг 17.2, *б*. Переход по гиперссылке из списка. Стандартный модуль

```
Private Sub ListBox1_DblClick(ByVal Cancel As MSForms.ReturnBoolean)
   Hyperlinks(ListBox1.ListIndex + 1).Follow
End Sub
```

Web-запрос и получение данных с Web-страницы

Web-страницы зачастую содержат информацию, пригодную для анализа в MS Excel. Например, в MS Excel можно анализировать котировки акций, используя данные, поступающие прямо с Web-страницы. При необходимости можно извлечь обновляемые данные (в этом случае их можно обновлять непосредственно в MS Excel в соответствии с последними изменениями Web-страницы) или получить данные с Web-страницы и хранить их на листе статически.

Использование Web-запроса позволяет получить с Web-страницы данные, например отдельную таблицу, несколько таблиц или весь текст, и провести их анализ, используя инструменты и свойства MS Excel. Щелкнув на соответствующей кнопке, можно обновить данные и получить с Web-страницы последние сведения. Например, можно получить и обновить котировки акций с общедоступной Web-страницы или таблицу с данными о продажах с Web-страницы организации.

Создание Web-запроса

Для создания Web-запроса необходимо произвести следующие действия:

1. Выберите команду **Данные | Импорт внешних данных | Создать веб-запрос**. На экране отобразится окно **Создание веб-запроса** (рис. 17.2).

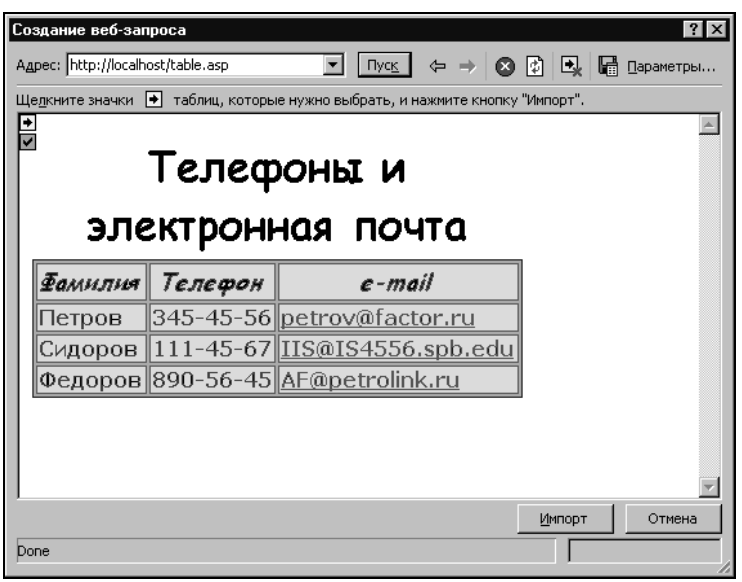


Рис. 17.2. Окно **Создание веб-запроса**

2. Поле **Адрес** предназначено для ввода URL искомой Web-страницы. Например, если на вашем компьютере установлен IIS, то его главным каталогом является C:\Inetpub\wwwroot. Расположите в каталоге C:\Inetpub \wwwroot файл Table.asp. В поле **Адрес** окна браузера нельзя указывать ссылку C:\Inetpub\wwwroot\table.asp. Вместо этого надо обратиться к файлу через сервер **http://localhost/table.asp**. Проще говоря, не забудьте воспользоваться своим сервером. Нажмите кнопку **Пуск**. В результате Web-страница Table.asp загрузится в окно **Создание веб-запроса**.

3. Используя кнопки со стрелками, выделите на Web-странице искомую таблицу.

4. Нажмите кнопку **Импорт**. Отобразится окно **Импорт данных** (рис. 17.3). Группа переключателей **Куда следует поместить данные?** состоит из двух возможностей: **Имеющийся лист** и **Новый лист**. Выберите переключатель **Имеющийся лист** и в соответствующее поле введите ссылку на ячейку (например **A1**), в которой будет располагаться верхний левый угол импортируемой таблицы.

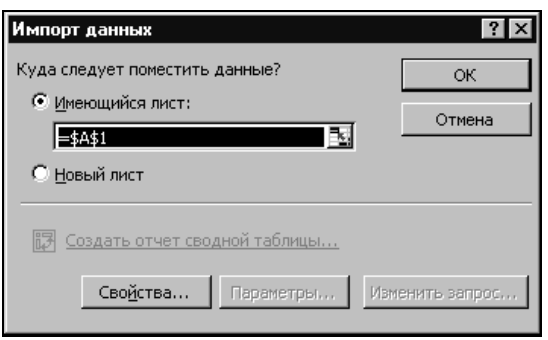


Рис. 17.3. Окно **Импорт данных**

5. Для того чтобы задать параметры запроса, нажмите кнопку **Свойства**. На экране отобразится окно **Свойства внешнего диапазона** (рис. 17.4). Поле **Имя** задает имя запроса. Группа **Определение запроса** устанавливает пароль и устанавливает, надо ли сохранять определение запроса. Группа **Обновление экрана** задает режим обновления импортированных данных. Группа **Формат и разметка данных** устанавливает параметры форматирования. Группа **Если количество строк в диапазоне изменится** задает алгоритм действия при изменении размеров диапазона. Флажок **заполнить формулами соседние столбцы** определяет, надо ли вводить вместе с данными и пояснительные формулы. Оставьте все предлагаемые по умолчанию параметры окна **Свойства внешнего диапазона**, и нажмите кнопку **OK**.

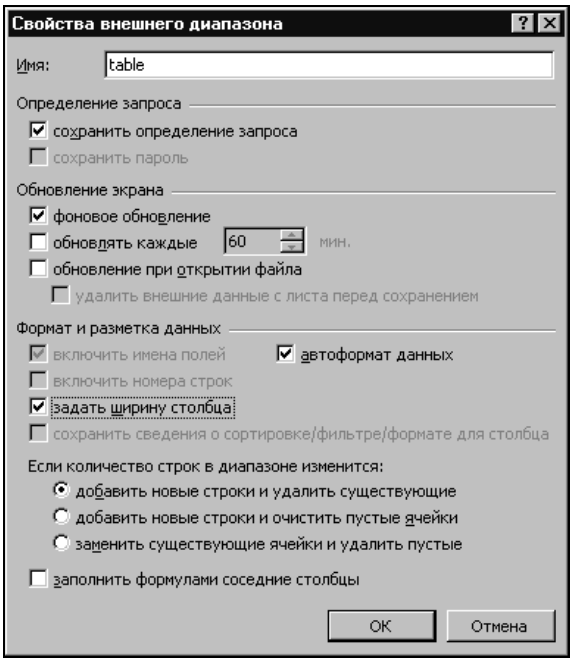
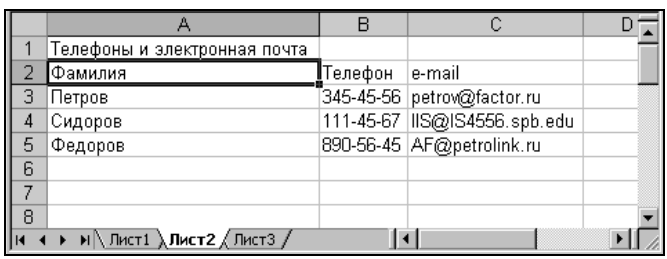


Рис. 17.4. Окно **Свойства внешнего диапазона**

Окно **Свойства внешнего диапазона** закроется. Нажмите кнопку **OK** окна **Импорт данных**. Данные из выбранной таблицы Web-страницы будут импортированы на рабочий лист (рис. 17.5).



	A	B	C	D
1	Телефоны и электронная почта			
2	Фамилия	Телефон	e-mail	
3	Петров	345-45-56	petrov@factor.ru	
4	Сидоров	111-45-67	IIS@IIS4556.spb.edu	
5	Федоров	890-56-45	AF@petrolink.ru	
6				
7				
8				

Лист1 Лист2 Лист3

Рис. 17.5. Таблица, импортированная на рабочий лист в результате Web-запроса

Создание скриптов для Web на стороне сервера

Скрипты Microsoft Active Service Pages (ASP, активные серверные страницы), написанные на VBA (точнее, на его урезанной части VBScript), позволяют создавать сценарии, которые проигрываются на стороне сервера. Под-

робно об HTML и ASP говорится в моей книге "Excel, VBA, Internet в экономике и финансах", опубликованной издательством "БХВ-Петербург". Из всей выходной информации ASP-сценарии передают на компьютер клиента только текст и HTML-теги, где они воспроизводятся в окне браузера. При помощи объектной модели ADO (ActiveX Data Object) ASP доступны базы данных, хранимые на сервере, а при помощи FSO — текстовые файлы, хранимые там же. Это позволяет легко создавать различные Web-приложения типа виртуальных каталогов. Для использования ASP необходимо установить Web-сервер корпорации Microsoft Internet Information Server (IIS) для работы с операционной системой Windows 2000 и Windows XP. Windows предоставляет возможность проигрывать серверные приложения на компьютере клиента. Для этого файл серверного приложения (например Day1.asp, приведенный в листинге 17.3) расположите в папке C:\Inetpub\wwwroot компьютера клиента. Это корневой каталог сервера, создаваемый по умолчанию при его инсталляции. Клиенты, подсоединенные к вашему компьютеру с помощью HTTP-протокола, автоматически попадают в этот каталог. В каталоге имеется файл Default.htm, который открывается в браузере клиента при его подсоединении к серверу без указания имени документа. В поле **Адрес** браузера введите адрес **http://127.0.0.1/day1.asp** или **http://localhost /day1.asp**, где **http://127.0.0.1** или **http://localhost** — зарезервированный IP-адрес для подключения к серверу, запущенному на компьютере, с которого поступил запрос на подключение.

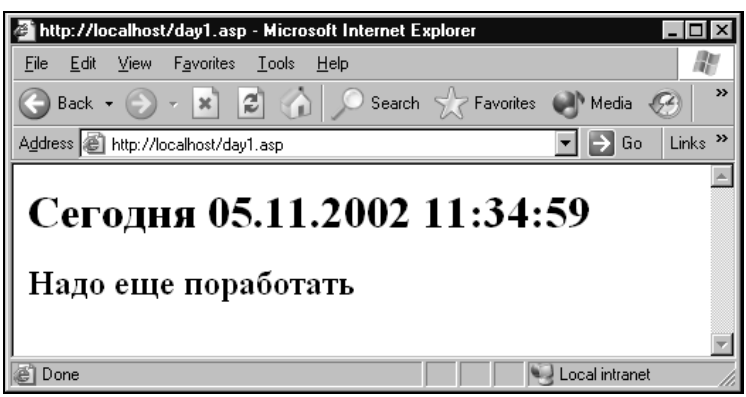


Рис. 17.6. Приветствие, передаваемое от сервера в зависимости от дня недели

В качестве примера создадим следующий код (листинг 17.3), отображающий в браузере различные приветствия в зависимости от дня недели, в который был загружен документ (рис. 17.6). Для того чтобы компьютер мог различать, какая часть кода из ASP-файла должна выполняться на серверной стороне, надо воспользоваться тегом `<% statements %>`, где *statements* — операторы, выполняемые на сервере. Знак равенства внутри тега `<%=Variable%>` представляет собой сокращенное обозначение метода `write` объекта

Response, который используется для пересылки информации от сервера клиенту.

Листинг 17.3. Ваш первый скрипт на стороне сервера. Файл day1.asp

```
<%@ Language=VBScript %>
<html>
<h1>Сегодня <%= Now%></h1>
<% If Weekday(Now) = vbFriday Then %>
<h2>Завтра начинаются выходные!!!!</h2>
<% ElseIf Weekday(Now) = vbSaturday Or  Weekday(Now) = vbSunday Then %>
<h2> Выходные!!!!</h2>
<% Else %>
<h2>Надо еще поработать</h2>
<% End If %>
</html>
```

Приведем тот же самый код, но написанный с использованием объекта Response и его метода write, который осуществляет пересылку информации от сервера клиенту (листинг 17.4).

Листинг 17.4. Ваш первый скрипт на стороне сервера. Файл day2.asp

```
<%@ Language=VBScript %>
<html>
<h1>Сегодня <%response.write Now%></h1>
<%
  If Weekday(Now) = vbFriday Then
     response.write "<h2>Завтра начинаются выходные!!!!</h2>"
  ElseIf Weekday(Now) = vbSaturday Or  Weekday(Now) = vbSunday Then
    response.write "<h2>Завтра начинаются выходные!!!!</h2>"
  Else
    response.write "<h2>Надо еще поработать</h2>"
  End If
%>
</html>
```

Передача данных от клиента серверу

Для передачи данных от клиента серверу часто используются формы, которые группируют в себе несколько элементов управления. В следующем примере (листинг 17.5) на форме расположено два поля: **Имя** и **E-Mail**, и две кнопки: **Submit** и **Reset** (рис. 17.7). Нажатие кнопки **Submit** инициирует

выполнение ASP-файла, указанного в качестве значения атрибута `action` тега `form` (в данном случае Answer.asp). Свойство `form` серверного объекта `Request` позволяет передать этому файлу значения из тегов управления, расположенных в форме. Идентификация тегов производится по значениям их атрибутов `name`. По заполнении формы пользователь нажимает кнопку **Submit**, что вызывает передачу данных на сервер и обработку их программой Answer.asp, которая подтверждает получение сервером данных (рис. 17.8). Нажатие же кнопки **Reset** приведет к переустановке значений элементов управления, входящих в форму, к тем, которые используются по умолчанию.

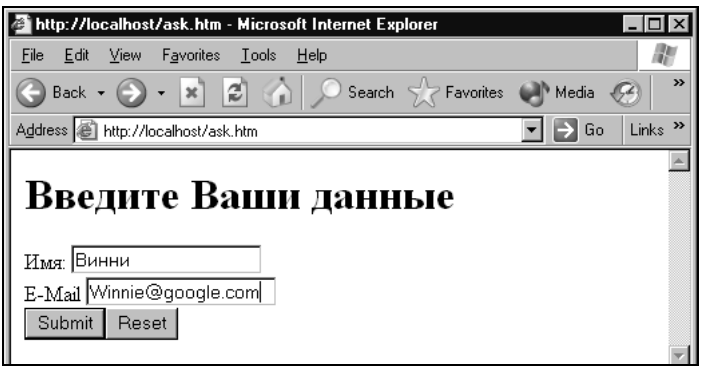


Рис. 17.7. Форма для передачи данных

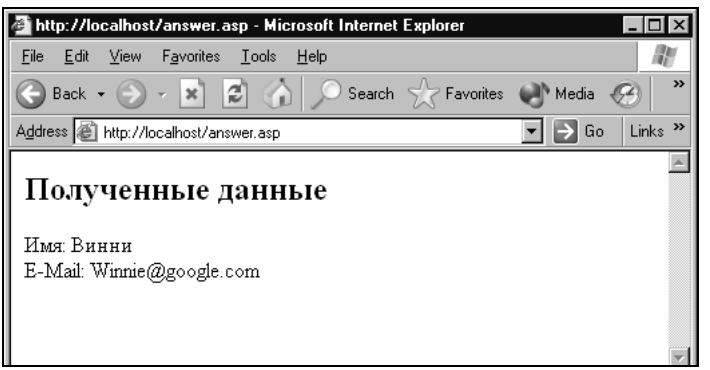


Рис. 17.8. Ответ, переданный сервером клиенту

Листинг 17.5, *а*. Передача данных от клиента серверу. Файл Ask.htm

```
<html>
<body>
<h1>Введите Ваши данные</h1>
```

```
<form id="frmName" method="post" action="http:\\localhost\answer.asp">
  Имя: <input type="text" name="txtName">
  <br>
  E-Mail <input type="text" name="txtEMail">
 <br>
  <input type="submit" value="Submit"><input type="reset" value="Reset">
</form>
</body>
</html>
```

Листинг 17.5, *б*. Передача данных от клиента серверу. Файл Answer.asp

```
<%@ Language=VBScript %>
<html>
<body>
<%
   Response.Write "<H2> Полученные данные </H2>"
   Response.Write "Имя:  " & Request.form("txtName")
   Response.Write "<BR>"
   Response.Write "E-Mail: " & Request.form("txtEMail")
%>
</body>
</html>
```

Создание калькулятора на базе MS Excel в клиент-серверных приложениях

На сервере может располагаться MS Excel, и тогда допустимо обращение к его объектной модели. Продемонстрируем, как это можно сделать, на примере создания функционального калькулятора (листинг 17.6). Итак, имеется клиентское приложение с кнопками **Submit** и **Reset** и полем для ввода арифметического выражения (рис. 17.9). Нажатие кнопки **Submit** вызывает передачу этого выражения на сервер, где при помощи функции `CreateObject` организуется доступ к объекту `Application`, а дальше остается воспользоваться методом `Evaluate` и передать найденный результат клиенту (рис. 17.10).

Листинг 17.6, *а*. Калькулятор на базе MS Excel в клиент-серверных приложениях. Файл Startcalc.asp

```
<%@ Language=VBScript %>
<html>
<body>
```

```
<h1>Введите арифметическое выражение</h1>
<p>
<form id="frmCalc" action="http:\\localhost\excelcalc.asp" method="post">
<input type="text" name="txtExpr">
<p>
<input type="submit" value="Submit">
<input type="reset" value="Reset">
</body>
</html>
```

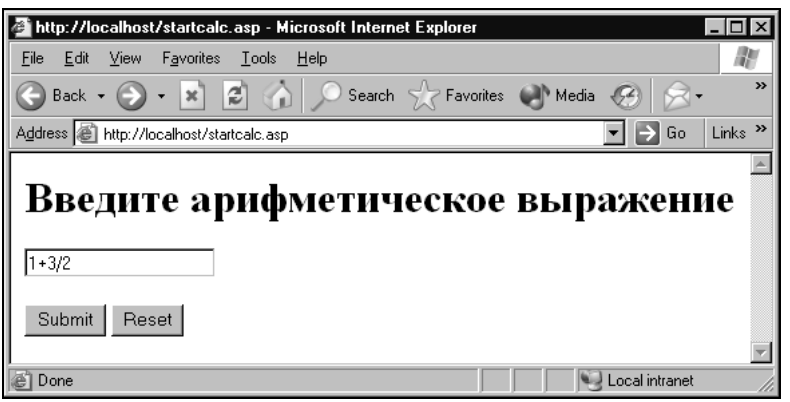


Рис. 17.9. Форма для передачи данных

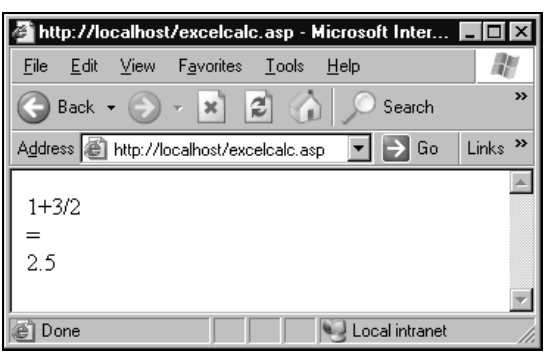


Рис. 17.10. Ответ, переданный сервером клиенту

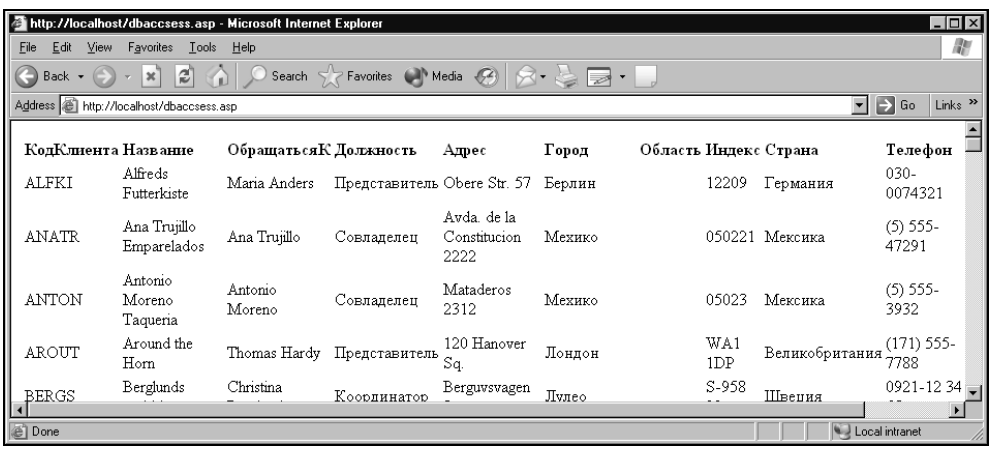
Листинг 17.6, *б*. Калькулятор на базе MS Excel в клиент-серверных приложениях. Файл Excelcalc.asp

```
<%
   Dim objExcel, sExpr
   Set objExcel = CreateObject("Excel.Application")
```

```
    sExpr = request.form("txtExpr")
    response.write sExpr
    response.write "<br> = <br>"
    response.write objExcel.Evaluate(sExpr)
    Set objExcel = Nothing
%>
```

Создание Web-страницы для доступа к базе данных, расположенной на сервере

Модель ADO позволяет производить доступ к базе данных, расположенной на сервере по той же схеме, как если бы она была расположена на стороне клиента. Следующий код (листинг 17.7) демонстрирует, как можно прочитать данные из таблицы **Клиенты** базы данных **Борей** и отобразить их в виде таблицы в окне браузера (рис. 17.11). Метод `Mappath` объекта `Server` переводит виртуальный путь в физический. Поэтому для функционирования данного приложения надо расположить файл Борей.mdb в каталоге C:\Inetpub\wwwroot.



КодКлиента	Название	ОбращатьсяК	Должность	Адрес	Город	Область	Индекс	Страна	Телефон
ALFKI	Alfreds Futterkiste	Maria Anders	Представитель	Obere Str. 57	Берлин		12209	Германия	030-0074321
ANATR	Ana Trujillo Emparelados	Ana Trujillo	Совладелец	Avda. de la Constitucion 2222	Мехико		050221	Мексика	(5) 555-47291
ANTON	Antonio Moreno Taqueria	Antonio Moreno	Совладелец	Mataderos 2312	Мехико		05023	Мексика	(5) 555-3932
AROUT	Around the Horn	Thomas Hardy	Представитель	120 Hanover Sq.	Лондон		WA1 1DP	Великобритания	(171) 555-7788
BERGS	Berglunds	Christina	Координатор	Berguvsvagen	Лулео		S-958	Швеция	0921-12 34

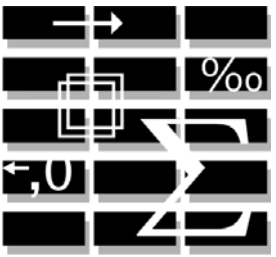
Рис. 17.11. Создание Web-страницы для доступа к базе данных, расположенной на сервере

Листинг 17.7. Создание Web-страницы для доступа к базе данных, расположенной на сервере

```
<%
  Dim cn
  Set cn = Server.CreateObject("ADODB.Connection")
  cn.Provider = "Microsoft.Jet.OLEDB.4.0"
```

```
  cn.open Server.Mappath("Борей.mdb")
  Dim sqlQuery
  sqlQuery = "SELECT * FROM Клиенты"
  Dim rs
  Set rs  = Server.CreateObject("ADODB.Recordset")
  Set rs = cn.execute(sqlQuery)
  Dim i
%>
<body>
<table>
   <tr>
     <% for i=0 to rs.fields.count-1 %>
     <td><b> <%= rs.fields(i).name %></b></td>
    <% next %>
   </tr>
   <%
      Do While Not rs.EOF
   %>
   <tr>
     <% for i=0 to rs.fields.count-1 %>
     <td> <%= rs.fields(i).value %></td>
    <% next %>
   </tr>
   <%
        rs.MoveNext
         Loop
         rs.Close
         cn.Close
         Set rs = Nothing
         Set cn = Nothing
   %>
</table>
</body>
</html>
```

ГЛАВА 18



MS Excel и Visual Basic .NET

В этой главе мы рассмотрим несколько примеров, как в проекты Visual Basic .NET можно инкапсулировать возможности MS Excel при разработке приложений, а именно: при создании калькуляторов, решении уравнений, построении графиков, конструировании отчетных таблиц. Для тех, кто интересуется Visual Basic .NET, рекомендую прочитать мою книгу "Visual Basic .NET: разработка приложений", выпущенную издательством "БХВ-Петербург". В ней очень подробно описаны как средства, так и техника создания проектов в .NET. Книга насыщена большим числом примеров (более 400), что существенно облегчает и ускоряет процесс изучения этой новой технологии программирования, созданной корпорацией Microsoft.

Открытие и создание рабочей книги

Приводимая далее программа показывает, как из кода можно открыть существующую и создать новую рабочую книгу MS Excel. Для реализации этого проекта создайте в интегрированной среде разработки приложений Visual Studio .NET форму, на которой расположите две кнопки. При помощи окна **Properties** установите значения свойств `Name` и `Text` кнопок равными `btnNew`, `Создать` и `btnOpen`, `Открыть`. Выберите команду **Project | Add Reference**. На экране отобразится окно **Add Reference**. В списке **COM** выберите **Microsoft Excel 10.0 Object Library**, нажмите кнопку **Select**. Выбранная ссылка добавится в список **Selected Components**, после чего нажмите кнопку **OK**, окно **Add Reference** автоматически закроется. Теперь можно переходить к набору программы. К коду, сгенерированному мастером проекта, добавьте следующие инструкции (листинг 18.1). Проект готов. Обратите внимание, как используется вспомогательная переменная `missing` типа `Object` — лишь для упоминания необязательных переменных, требуемых по синтаксису, без присваивания им значений.

Листинг 18.1. Открытие и создание рабочей книги

```
Private objExcel As Excel.Application
Private workbook As Excel.Workbook

Private Sub btnOpen_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles btnOpen.Click
   Dim ofd As New OpenFileDialog()
   objExcel = New Excel.Application()
   ofd.FileName = "*.xls"
   If (ofd.ShowDialog() = DialogResult.OK) Then
      Dim missing As Object = System.Reflection.Missing.Value
      workbook = objExcel.Workbooks.Open(fileName:=ofd.FileName, _
            UpdateLinks:=missing, ReadOnly:=missing, Format:=missing, _
            Password:=missing, WriteResPassword:=missing, _
            IgnoreReadOnlyRecommended:=missing, Origin:=missing, _
            Delimiter:=missing, Editable:=missing, Notify:=missing, _
            Converter:=missing, AddToMRU:=missing)
      objExcel.Visible = True
   End If
End Sub

Private Sub btnNew_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles btnNew.Click
   objExcel = New Excel.Application()
   workbook = objExcel.Workbooks.Add()
   workbook.ActiveSheet.Range("A1").Value = "Книга создана"
   objExcel.Visible = True
End Sub
```

Нахождение значений арифметических выражений

В качестве первого примера использования MS Excel для создания на Visual Basic .NET Windows-приложений рассмотрим приложение, которое по введенному в поле ввода арифметическому выражению вычисляет его значение. Например, в поле **Выражение** пользователь вводит арифметическое выражение `(1+4)/2+4`, нажимает кнопку **OK**, и значение выражения выводится в поле ввода `Результат` (рис. 18.1).

Перейдем непосредственно к конструированию приложения. Создайте форму с двумя полями ввода, двумя надписями и кнопкой. Используя окно

Properties, установите значения свойств элементов управления, перечисленные в табл. 18.1.

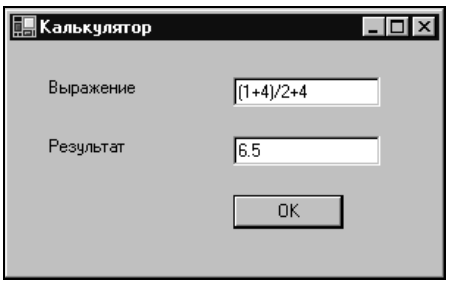


Рис. 18.1. Окно **Калькулятор**

Таблица 18.1. *Значения свойств, установленные в окне* ***Properties***

Элемент управления	Свойство	Значение
Надпись	Text	Выражение
Поле ввода	Name	txtExpr
	Text	""
Надпись	Text	Результат
Поле ввода	Name	txtRes
	Text	""
Кнопка	Name	OK
	Text	btnOK

Выберите команду **Project | Add Reference**. На экране отобразится окно **Add Reference**. В списке **COM** выберите **Microsoft Excel 10.0 Object Library**, нажмите кнопку **Select**. Выбранная ссылка добавится в список **Selected Components**, после чего нажмите кнопку **OK**, окно **Add Reference** автоматически закроется. Теперь можно переходить к набору программы. К коду, сгенерированному мастером проекта, добавьте следующие инструкции (листинг 18.2). Проект готов.

Листинг 18.2. Нахождение значений арифметических выражений

```
Private Sub btnOK_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles btnOK.Click
   Dim objExcel As Excel.Application = Nothing
   Dim expr As String
```

```
   Dim res As Double
   Try
      expr = txtExpr.Text.Trim
      expr = "=" & expr
      objExcel = New Excel.Application()
      If objExcel Is Nothing Then
         MessageBox.Show("Excel не открылся")
         Exit Sub
      End If
      res = objExcel.Evaluate(expr)
      txtRes.Text = res.ToString
      objExcel.Quit()
   Catch ex As Exception
      MessageBox.Show("Ошибка в выражении")
   End Try
End Sub
```

Проверка орфографии

Воспользуемся MS Excel для проверки орфографии. В данном демонстрационном проекте в поле ввода пользователь вводит какой-либо текст, нажимает кнопку **Проверка**. При помощи средств MS Excel происходит проверка введенного текста. В случае нахождения орфографической ошибки на экране отображается встроенное диалоговое окно, предлагающее варианты исправления. По завершении проверки в поле ввода отображается исправленное предложение (рис. 18.2).

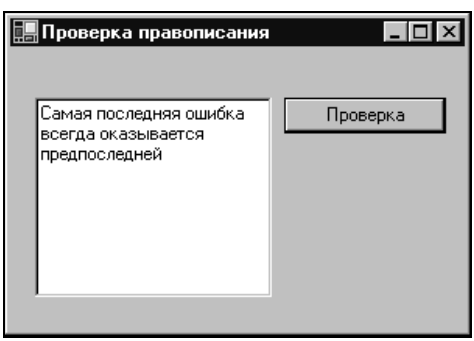


Рис. 18.2. Проверка орфографии

Для конструирования данного предложения создайте форму, на которой расположите поле ввода и кнопку. Используя окно **Properties**, установите значения свойств элементов управления, перечисленные в табл. 18.2.

Таблица 18.2. *Значения свойств, установленные в окне* ***Properties***

Элемент управления	Свойство	Значение
Поле ввода	Name	txtSentence
	Text	""
	Multiline	True
Кнопка	Name	Проверка
	Text	btnSpelling

Выберите команду **Project | Add Reference**. На экране отобразится окно **Add Reference**. В списке **COM** выберите **Microsoft Excel 10.0 Object Library**, нажмите кнопку **Select**. Выбранная ссылка добавится в список **Selected Components**, после чего нажмите кнопку **OK**, окно **Add Reference** автоматически закроется. Теперь можно переходить к набору программы. К коду, сгенерированному мастером проекта, добавьте следующие инструкции (листинг 18.3). Проект готов.

Листинг 18.3. Проверка орфографии

```
Private Sub btnSpelling_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles btnSpelling.Click
   Dim objExcel As Excel.Application
   objExcel = New Excel.Application()
   With objExcel
      .Workbooks.Add()
      .Visible = False
      .DisplayAlerts = False
   End With
   With objExcel.ActiveSheet.Cells(1, 1)
      .Value = txtSentence.Text
      .CheckSpelling()
      txtSentence.Text = .Value
   End With
   objExcel.Quit()
   objExcel = Nothing
End Sub
```

Решение уравнения

Для решения уравнений Excel обладает мощным средством — командой **Подбор параметра**. Она позволяет на рабочем листе найти корень уравнения, но использование подбора параметра при решении уравнения требует выполнения относительно большого объема работы:

1. Представить уравнение в таком виде, чтобы неизвестное входило только в его левую часть, а правая часть была постоянной. Например, уравнение $x^2 = x + 1$ надо привести к виду $x^2 - x - 1 = 0$.

2. Выбрать команду **Сервис | Параметры** и на вкладке **Вычисления** появившегося диалогового окна **Параметры** в поле **Относительная погрешность** указать погрешность, с которой будет находиться корень уравнения. Например, 10^{-5}.

3. На рабочем листе отвести одну ячейку, например **B1**, под неизвестное. Ввести в эту ячейку начальное приближение к корню, например, 1.

4. Отвести другую ячейку, например **B2**, на рабочем листе под левую часть уравнения. Ввести в эту ячейку формулу левой части уравнения. В рассматриваемом случае формулу `=B1^2-B1-1`.

5. Выбрать команду **Сервис | Подбор параметра** и в появившемся диалоговом окне **Подбор параметра** в поле **Установить в ячейке** дать ссылку на ячейку, отведенную под левую часть уравнения. В нашем конкретном случае, **B2**. В поле **Значение** ввести число, стоящее в правой части уравнения. В нашем случае надо ввести `0`. В поле **Изменяя значение ячейки** дать ссылку на ячейку, отведенную под неизвестное, в рассматриваемом примере **B1**. Нажать кнопку **OK**.

6. На экране отобразится диалоговое окно **Результат подбора параметра**, а в ячейке **B1** вместо начального приближения будет находиться найденное значение корня. В данном случае 1.618034.

Как можно видеть, применение команды **Подбор параметра** достаточно трудоемко. Давайте создадим на Visual Basic .NET Windows-приложение, облегчающее работу пользователя при решении уравнения. Создайте форму с четырьмя надписями, четырьмя полями ввода, вертикальной полосой прокрутки и кнопкой (рис. 18.3) и при помощи окна **Properties** установите им значения свойств, как показано в табл. 18.3.

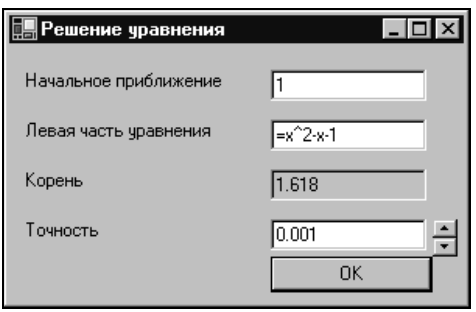


Рис. 18.3. Окно **Решение уравнения**

Таблица 18.3. *Значения свойств, установленные в окне* ***Properties***

Элемент управления	Свойство	Значение
Надпись	Text	Начальное приближение
Поле ввода	Name	txtInit
	Text	""
Надпись	Text	Левая часть уравнения
Поле ввода	Name	txtEquation
	Text	""
Надпись	Text	Корень
Поле ввода	Name	txtRoot
	Text	""
Надпись	Text	Точность
Поле ввода	Name	txtTol
	Text	""
Полоса прокрутки	Name	vsbTol
Кнопка	Name	btnOK
	Text	OK

Опишем в табл. 18.4 предназначение каждого элемента управления данного проекта.

Таблица 18.4. *Предназначение элементов управления*

Элемент управления	Описание
Поле ввода **Начальное приближение**	Пользователь вводит начальное приближение к корню
Поле ввода **Левая часть уравнения**	Пользователь вводит левую часть уравнения. Уравнение должно быть приведено к виду, когда его правая часть равна 0. Вводимая левая часть уравнения должна начинаться со знака равенства, записываться по правилам языка программирования и вместо неизвестного должен быть использован символ x. Например: =x^2-x-1 В программе введенное выражение будет преобразовано в формулу рабочего листа

Таблица 18.4 *(окончание)*

Элемент управления	Описание
Поле ввода **Корень**	Выводится найденное значение корня. Поле недоступно для пользователя
Поле ввода **Точность**	Вводится относительная точность нахождения корня при помощи полосы прокрутки
Полоса прокрутки	Задает относительную точность нахождения корня, которая выводится в поле `txtTol`
Кнопка	Считывает начальное приближение, левую часть уравнения и точность нахождения корня. Присваивает ячейке **B1** активного рабочего листа имя `x`. Проверяет, является ли введенное начальное приближение числом, и преобразуется ли введенная левая часть уравнения в формулу рабочего листа. Если нет, то на экране отображается сообщение, и выполнение программы прерывается. Если данные введены корректно, то начальное приближение вводится в ячейку **B1** (ее имя теперь `x`) рабочего листа. Формула левой части уравнения вводится в ячейку **B2**. При помощи метода **Подбор параметра** (метод `GoalSeek` диапазона) ищется корень. Если корень не найден, отображается сообщение. Если корень найден, то он предварительно форматируется с учетом введенной точности нахождения, а затем выводится в поле `txtRoot`

Выберите команду **Project | Add Reference**. На экране отобразится окно **Add Reference**. В списке **COM** выберите **Microsoft Excel 10.0 Object Library**, нажмите кнопку **Select**. Выбранная ссылка добавится в список **Selected Components**, после чего нажмите кнопку **OK**, окно **Add Reference** автоматически закроется. Теперь можно переходить к набору программы. К коду, сгенерированному мастером проекта, добавьте следующие инструкции (листинг 18.4). Проект готов.

Листинг 18.4. Решение уравнения

```
Private objExcel As Excel.Application
Private objSheet As Excel.Worksheet

Private Sub Form1_Load(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles MyBase.Load
   objExcel = New Excel.Application()
```

```
   With objExcel
      .Workbooks.Add()
      .ActiveSheet.Name = "Уравнение"
      .Visible = False
      .DisplayAlerts = False
   End With
   objSheet = objExcel.Worksheets("Уравнение")
   txtRoot.ReadOnly = True
   txtTol.ReadOnly = True
   With vsbTol
      .Minimum = 1
      .Maximum = 7
      .LargeChange = 1
      .SmallChange = 1
      .Value = 3
   End With
End Sub

Private Sub btnOK_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles btnOK.Click
   Dim Equation As String ' Уравнение, например, =x^2-x-1
   Dim Guess As Double ' Начальное приближение
   Dim Root As Double  ' Корень
   Dim Test As Boolean ' Параметр, информирующий о том, найден ли корень
   Try
      Guess = CDbl(txtInit.Text)
   Catch ex As Exception
      MessageBox.Show("Корректно введите начальное приближение")
      txtInit.Focus()
      Exit Sub
   End Try
   objSheet.Range("B1").Value = Guess
   ' Присваивает ячейке B1 имя "x"
   objSheet.Range("B1").Name = "x"

   Equation = txtEquation.Text.Trim.ToLower
   objSheet.Range("B2").Value = Equation
   Try
      Dim d As Double = CDbl(objSheet.Range("B2").Value)
   Catch ex As Exception
      MessageBox.Show("Корректно введите функцию")
      txtEquation.Focus()
      Exit Sub
   End Try
```

```
   ' Нахождение корня Подбором параметра
   Test = objSheet.Range("B2").GoalSeek(Goal:=0, _
                                ChangingCell:=objSheet.Range("x"))
   If Not Test Then
      MessageBox.Show("Корень не найден")
      Exit Sub
   End If
   Root = objSheet.Range("x").Value
   ' Создание пользовательского формата с учетом
   ' заданной точности нахождения корня
   Dim UserFormat As String = "#."
   Dim I As Integer
   For I = 1 To vsbTol.Value
      UserFormat += "0"
   Next
   Root = Format(Root, UserFormat)
   txtRoot.Text = Root
End Sub

Private Sub vsbTol_ValueChanged(ByVal sender As Object, _
      ByVal e As System.EventArgs) Handles vsbTol.ValueChanged
   txtTol.Text = 0.1 ^ vsbTol.Value
   objExcel.MaxChange = CDbl(txtTol.Text)
End Sub

Private Sub Form1_Closing(ByVal sender As Object, _
      ByVal e As System.ComponentModel.CancelEventArgs) _
      Handles MyBase.Closing
   objExcel.Quit()
   objSheet = Nothing
   objExcel = Nothing
End Sub
```

Построение графика

Сконструируем в Visual Basic .NET приложение, которое позволяет построить график функции. В поля ввода диалогового окна приложения **Построение графика** (рис. 18.4) надо ввести формулу функции, начальное и конечное значение интервала, на котором строится функция, а также шаг, с которым вычисляются последовательные значения функции при построении ее графика. Перейдем непосредственно к конструированию приложения. Создайте форму с четырьмя полями ввода и четырьмя надписями, рисунком и кнопкой. Установите значения свойств элементов управления при помощи окна **Properties**, как показано в табл. 18.5.

Таблица 18.5. *Значения свойств, установленные в окне* ***Properties***

Элемент управления	Свойство	Значение
Надпись	Text	Начальное значение
Поле ввода	Name	txtBegin
Надпись	Text	Шаг
Поле ввода	Name	txtStep
Надпись	Text	Конечное значение
Поле ввода	Name	txtEnd
Надпись	Text	Функция
Поле ввода	Name	txtFun
Графическое поле	Name	picGraph
Кнопка	Name	btnOK
	Text	OK

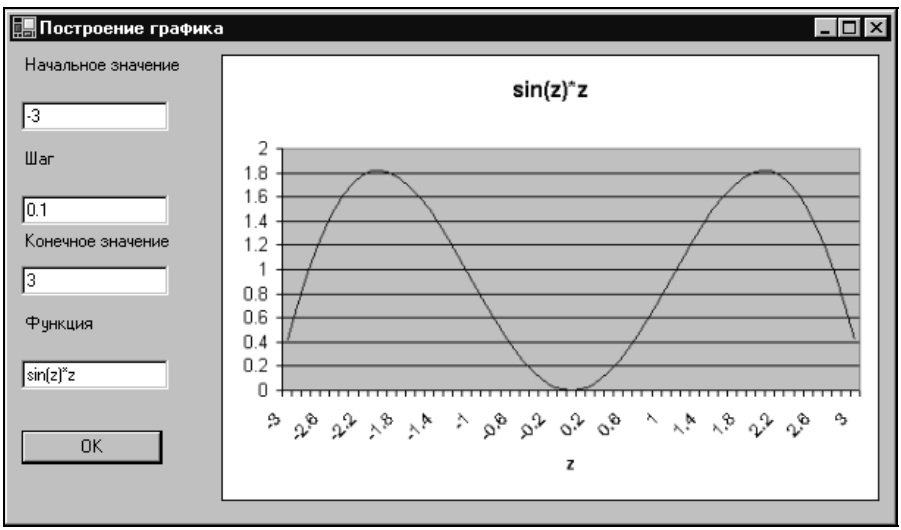


Рис. 18.4. Окно **Построение графика**

Выберите команду **Project | Add Reference**. На экране отобразится окно **Add Reference**. В списке **COM** выберите **Microsoft Excel 10.0 Object Library**, нажмите кнопку **Select**. Выбранная ссылка добавится в список **Selected Components**, после чего нажмите кнопку **OK**, окно **Add Reference** автоматически закроется. Теперь можно переходить к набору программы. К коду, сгенери-

рованному мастером проекта, добавьте инструкции из листинга 18.5. Проект готов.

Примечание

Для сокращения кода в качестве параметра функции, которая вводится в поле ввода **Функция**, надо использовать символ `z`. Например, в поле ввода **Функция** можно ввести `sin(z)*z` или `z/(1+z^2)`. Данная программа создает объект — рабочую книгу MS Excel и в ней рабочий лист **Graph**. На рабочем листе по введенным параметрам задачи строится график, графический образ которого записывается в растровый файл Tempgraph.gif, а потом из этого файла растровая картинка загружается в графическое поле.

Листинг 18.5. Построение графика

```
Private Sub Form1_Load(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles MyBase.Load
   txtBegin.Clear()
   txtStep.Clear()
   txtEnd.Clear()
   txtFun.Clear()
   picGraph.SizeMode = PictureBoxSizeMode.StretchImage
End Sub

Private Sub btnOK_Click(ByVal sender As System.Object, _
     ByVal e As System.EventArgs) Handles btnOK.Click
   Dim objExcel As Excel.Application
   Dim a As Double, b As Double, h As Double
   Dim f As String
   Dim fileName As String = "c:\tempgraph.gif"
   Dim rngFun As Excel.Range
   Dim n As Long
   Const xlColumns = 2
   Const xlLine = 4
   Const xlCategory = 1
   Const xlPrimary = 1
   Const xlLocationAsObject = 2

   Try
      a = CDbl(txtBegin.Text)
      h = CDbl(txtStep.Text)
      b = CDbl(txtEnd.Text)
   Catch
      MessageBox.Show("Ошибка в начальных данных")
      Exit Sub
   End Try
```

```
If (b — a) / h < 1 Then
   MessageBox.Show("Начальные данные не согласованы")
   Exit Sub
End If

f = Trim(txtFun.Text)
objExcel = New Excel.Application()
With objExcel
   .Workbooks.Add()
   .ActiveSheet.Name = "Graph"
   .Visible = False
   .DisplayAlerts = False
End With
f = f.ToLower()
f = f.Replace("z", "A2")
f = "=" & f

With objExcel.Worksheets("Graph")
   .Range("A2").Value = a
   .Range("A2").DataSeries(Rowcol:=xlColumns, Step:=h, Stop:=b)
   n = objExcel.WorksheetFunction.CountA(.Columns(1)) + 1
   .Range("B2").Value = f
   rngFun = .Range(.Cells(2, 2), .Cells(n, 2))
   .Range("B2").AutoFill(Destination:=rngFun)
End With

With objExcel
   .Charts.Add()
   With .ActiveChart
      .ChartType = xlLine
      .SetSourceData(Source:=rngFun, PlotBy:=xlColumns)
      .SeriesCollection(1).XValues = _
                     "=Graph!R2C1:R" & n.ToString & "C1"
      .Location(Where:=xlLocationAsObject, Name:="Graph")
   End With
   With .ActiveChart
      .HasTitle = True
      .ChartTitle.Characters.Text = Trim(txtFun.Text)
      .Axes(xlCategory, xlPrimary).HasTitle = True
      .Axes(xlCategory, xlPrimary).AxisTitle.Characters.Text = "z"
      .HasLegend = False
      .Export(fileName)
   End With
```

```
        picGraph.Image = Image.FromFile(fileName)
    End With
    objExcel.Quit()
    objExcel = Nothing
End Sub
```

Создание отчета на рабочем листе

Продемонстрируем, как можно построить отчетный документ MS Excel на основе данных, отображаемых в списке (рис. 18.5).

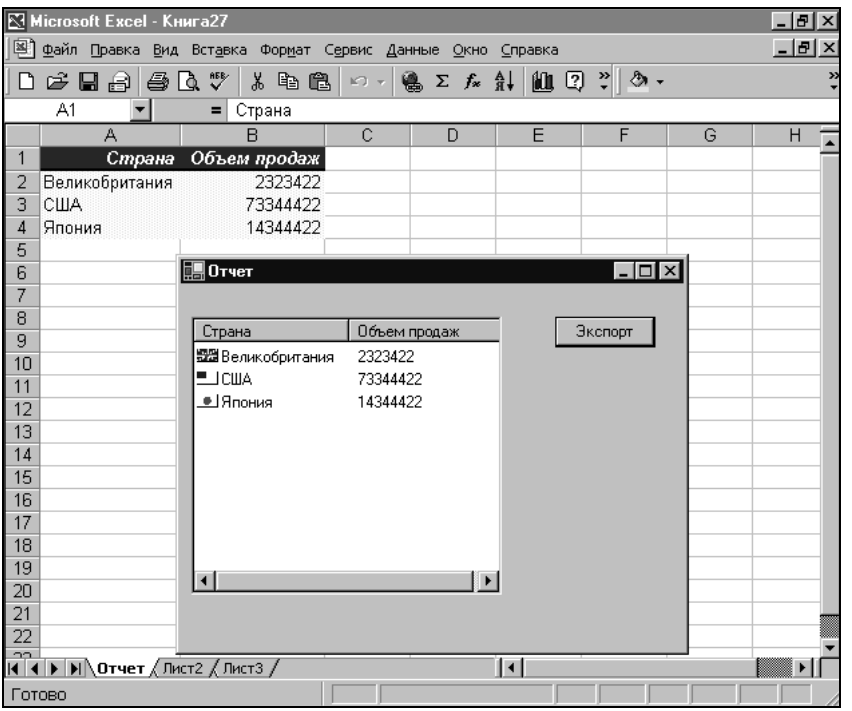


Рис. 18.5. Создание отчета на рабочем листе

Приводимая ниже программа создает новую рабочую книгу, в которой на основе данных из списка конструируется таблица. Для реализации этого проекта создайте форму, на которой расположите элемент управления **ListView** и кнопку. При помощи окна **Properties** установите значения свойств `Name` и `Text` кнопки равными `btnExport` и `Экспорт`. Выберите команду **Project | Add Reference**. На экране отобразится окно **Add Reference**. В списке **COM** выберите **Microsoft Excel 10.0 Object Library**, нажмите кнопку **Select**. Выбранная ссылка добавится в список **Selected Components**, после чего на-

жмите кнопку **OK**, окно **Add Reference** автоматически закроется. Теперь можно переходить к набору программы. К коду, сгенерированному мастером проекта, добавьте следующие инструкции (листинг 18.6). Проект готов.

Листинг 18.6. Создание отчета в Excel

```
Private objExcel As Excel.Application

Private Sub Form1_Load(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles MyBase.Load
   Dim imgLst As New ImageList()
   imgLst.Images.Add(New Icon("c:\flags\Flguk.ico"))
   imgLst.Images.Add(New Icon("c:\flags\Flgusa02.ico"))
   imgLst.Images.Add(New Icon("c:\flags\Flgjapan.ico"))

   ListView1.View = View.Details
   ListView1.SmallImageList = imgLst
   ListView1.Columns.Add("Страна", ListView1.Width / 2, _
      HorizontalAlignment.Left)
   ListView1.Columns.Add("Объем продаж", ListView1.Width / 2, _
      HorizontalAlignment.Left)
   Dim li(2) As String
   li(0) = "Великобритания"
   li(1) = "2323422"
   ListView1.Items.Add(New ListViewItem(li, 0))
   li(0) = "США"
   li(1) = "73344422"
   ListView1.Items.Add(New ListViewItem(li, 1))
   li(0) = "Япония"
   li(1) = "14344422"
   ListView1.Items.Add(New ListViewItem(li, 2))
End Sub

Private Sub btnExport_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles btnExport.Click
   Dim objSheet As Excel.Worksheet
   objExcel = New Excel.Application()
   With objExcel
      .Workbooks.Add()
      .ActiveSheet.Name = "Отчет"
      .Visible = True
      .DisplayAlerts = False
   End With
```

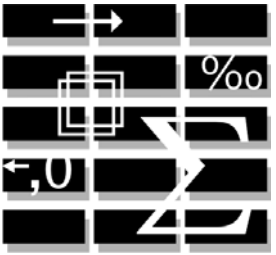
```
   objSheet = objExcel.Worksheets("Отчет")
   Dim i As Integer, j As Integer
   Dim col As Integer = ListView1.Columns.Count
   Dim row As Integer = ListView1.Items.Item(0).SubItems.Count

   For i = 1 To col
      objSheet.Cells(1, i).Value = ListView1.Columns.Item(i - 1).Text
   Next

   For i = 1 To col
      For j = 1 To row
         objSheet.Cells(i + 1, j).Value _
            = ListView1.Items.Item(i - 1).SubItems(j - 1).Text
      Next
   Next
   Const xlRangeAutoFormatColor2 As Integer = 8
   objSheet.Range(objSheet.Cells(1, 1), _
            objSheet.Cells(row + 1, col)).AutoFormat(_
            Format:=xlRangeAutoFormatColor2, Number:=True, Font:=True, _
            Alignment:=True, Border:=True, Pattern:=True, Width:=True)
End Sub

Private Sub Form1_Closing(ByVal sender As Object, _
      ByVal e As System.ComponentModel.CancelEventArgs) _
   Handles MyBase.Closing
   If Not (objExcel Is Nothing) Then
      objExcel.Quit()
      objExcel = Nothing
   End If
End Sub
```

ГЛАВА 19



MS Excel и XML

Расширенный язык разметки (Extensible Markup Language, XML) был создан в 1996 году Консорциумом W3C (World Wide Web Consortium) как *мета-язык*, предназначенный для описания языков разметки. В языках разметки, подобным HTML, имеются *теги* для структурирования данных. О языке HTML вы можете прочитать в моей книге "Excel, VBA и Internet в экономике и финансах", опубликованной издательством "БХВ-Петербург". Многие называют XML заменой HTML, но это совсем не так. В то время как HTML содержит фиксированный набор тегов, в XML теги вообще отсутствуют. Вместо этого XML позволяет разработчику самому создавать такой язык разметки, который в точности соответствует требованиям конкретного приложения. В приложениях XML обычно используются следующие данные:

- сам файл XML, имеющий строго определенную структуру;
- определение типа документа (Document Type Definition, DTD), в котором задается структура XML-файла. DTD является необязательной частью данных;
- таблица стилей (eXtensible Stylesheet Language, XSL), которая содержит информацию о том, как данные должны быть отформатированы при выводе. XSL является необязательной частью данных;
- процессор XML, т. е. программный модуль, обеспечивающий доступ приложения к данным, хранящихся в XML-документах;
- различные служебные функции для манипулирования данными и их переформатированием.

Просмотр XML-файлов в MS Excel

MS Excel позволяет просматривать любые XML-файлы. Для этого:

1. Выберите команду **Файл | Открыть**. Отобразится окно **Открытие документа**.

2. В списке **Тип файлов** выберите **Файлы XML**. Выберите искомый файл и нажмите кнопку **Открыть**.

Корпорация Microsoft создала специальный XML-формат (Spreadsheet, XMLSS), в котором при помощи команды **Файл | Сохранить как** можно сохранить рабочую книгу.

Синтаксис XML

Синтаксис XML максимально прост. Предположим, что вы менеджер фирмы Seekoei по выращиванию гиппопотамов, и вам надо на Web-сайте привести самую последнюю информацию о сотрудниках фирмы. В листинге 19.1 приводится часть HTML-кода, содержащего искомые данные.

Листинг 19.1. HTML-код, отображающий таблицу с искомыми данными

```
<table>
   <tr>
       <th>LastName </th>
       <th>FirstName</th>
       <th>E-mail</th>
   </tr>
   <tr>
       <td>Bond </td>
       <td>James</td>
       <td>bond007@yandex.com</td>
   </tr>
   <tr>
       <td>Cooper</td>
       <td>Gary</td>
       <td>gcooper@yandex.com</td>
   </tr>
</table>
```

HTML-код только определяет, как должен быть отформатирован текст. Автоматическое же извлечение информации из статической HTML-страницы оказывается весьма трудной задачей. Конечно, для того чтобы узнать электронные адреса сотрудников, вы можете написать программу, которая извлекает данные из третьего столбца таблицы, но при изменении ее дизайна эта программа перестанет работать.

В листинге 19.2 показано, как та же информация может быть представлена с помощью XML.

Листинг 19.2. XML, представляющий таблицу с искомыми данными

```
<?xml version="1.0" standalone="yes" ?>
<employee>
    <person>
        <lastname>Bond</lastname>
        <firstname>James</firstname>
        <email>bond007@yandex.com</email>
    </person>
    <person>
        <lastname>Cooper</lastname>
        <firstname>Gary</firstname>
        <email>gcooper@yandex.com</email>
     </person>
</employee>
```

Первая строка листинга — это объявление XML, которое содержит информацию для анализатора XML. Эта строка не является обязательной, но, как правило, она присутствует. Атрибут `version="1.0"` указывает версию XML. Атрибут `standalone="yes"` сообщает, что у данного документа отсутствует определение DTD, которое задает структуру документов и, в частности, названия элементов. Если имя элемента не определено в DTD, то он не может использоваться в данном XML-документе. Наличие DTD у документа не является обязательным, как это и происходит в данном примере.

Следом за объявлением XML идут его элементы. Элементы состоят из открывающих и закрывающих тегов, например `<lastname>...</lastname>`. Документ XML имеет иерархическую древовидную структуру, в вершине которой находится единственный элемент, называемый *корневым*. В данном примере таковым является элемент `<employee>`. Он включает в себя все остальные элементы. Текст, заключенный между открывающим и закрывающим тегами элемента, называется его *содержанием*. Например, содержанием элемента `<product>` является `MS Excel 2002`.

```
<product>MS Excel 2002</product>
```

Содержание относится к символьным данным, а элементы — к разметке документа. В свою очередь, символьные данные подразделяются на проверяемые символьные данные (Parsed Character Data, PCDATA) и не проверяемые (Unparsed Character Data).

Элементы могут не иметь содержания. Например:

```
<cellphone></cellphone>
```

В этом случае их можно объединять в один тег `<cellphone/>`.

У элементов могут быть атрибуты. Атрибут — это пара: *имя=значение*, расположенная в открывающем теге. Например, `href` является атрибутом тега `<a>`.

```
<a href="www.bhv.ru">
```

Атрибуты упрощают жизнь и укорачивают код. Например, элемент `<person>` можно было бы с помощью атрибутов записать следующим образом (листинг 19.3).

Листинг 19.3. Атрибуты сокращают код

```
<employee>
   <person lastname="Bond" firstname="James"
           email="bond007@yandex.com"/>
   <person lastname="Cooper " firstname="Gary"
           email="gcooper@yandex.com"/>
</employee>
```

Кроме того, атрибуты позволяют разбивать элементы на категории. Например, в следующем коде (листинг 19.4) в зависимости от значения атрибута `type` в элементе `<person>` содержится либо конфиденциальная, либо общедоступная информация.

Листинг 19.4. Разбивка элементов по категориям при помощи атрибутов

```
<?xml version="1.0" standalone="yes" ?>
<employee>
   <person type="work">
      <lastname>Bond</lastname>
      <firstname>James</firstname>
      <email>bond007@yandex.com</email>
   </person>
   <person type="work">
      <lastname>Cooper</lastname>
      <firstname>Gary</firstname>
      <email>gcooper@yandex.com</email>
   </person>
   <person type="personal">
      <lastname>Cooper</lastname>
      <firstname>Gary</firstname>
      <marriedstatus>new married</marriedstatus>
      <homephone>354-56-56</homephone>
   </person>
</employee>
```

Русификация XML

Для русификации XML при объявлении документа надо дополнительно указать атрибут `encoding`, задающий его кодировку (листинг 19.5).

Листинг 19.5. Русификация XML

```
<?xml version="1.0" encoding="koi8-r"?>
<employee>
   <person firstname="Джеймс " email="bond007@yandex.com">
      Бонд
   </person>
</employee>
```

Применение каскадной таблицы стилей для форматного вывода текста

Для форматного вывода текста в окно браузера можно использовать каскадную таблицу стилей (CSS). Для этого в XML надо добавить еще одно объявление, указывающее ссылку на файл (в данном случае Video.css) с используемой каскадной таблицей стилей (листинг 19.6).

```
<?xml:stylesheet href="video.css" type="text/css"?>
```

О каскадной таблице стилей можно прочитать, например, в моей книге "Excel, VBA, Internet в экономике и финансах", опубликованной издательством "БХВ-Петербург".

Листинг 19.6, *а*. Файл Video.xml

```
<?xml version="1.0"?>
<?xml:stylesheet href="video.css" type="text/css"?>
<videolib>
   <video>
    <title>Bridget Jones's Diary</title>
    <artists>Colin Firth, Hugh Grand and Renee Zellweger</artists>
    <impresion>Terrific fun</impresion>
   </video>
   <video>
      <title>Dirty Rotten Scoundrels</title>
      <artists>Steve Martin, Michael Caine</artists>
      <impresion>A delight!</impresion>
    </video>
</videolib>
```

Листинг 19.6, *б*. Файл Video.css

```
title
{
   display: block;
   font-family: Arial, Helvetica;
   font-size: 20pt;
   color: #c71585;
   text-align: left;
}
artists
{
   display: block;
   font-family: Arial, Helvetica;
   font-weight:bold;
   font-size: 20pt;
   color: #9370db;
   text-align: center;
}
impresion
{
   diplay:block;
   font-family: Arial, Helvetica;
   font-size: 16pt;
   color: #9370db;
   text-align: center;
}
```

Комментарии в XML

Комментарии в XML имеют тот же вид, что и в HTML. Комментарии могут располагаться в любом месте документа, кроме разметки.

```
<!- Здесь идет комментарий -->
```

Что такое DTD

DTD определяет элементы, используемые в документе. Он также задает структурные связи между элементами, когда один из них содержится внутри другого, а также их атрибуты. Может задать, какие атрибуты являются обязательными, а какие — нет.

DTD — это запись правил, чтобы процессор мог проверить соответствие документа его собственному набору правил. Если это соответствие наруше-

но, то документ является недопустимым. В общем случае, документ XML не обязан иметь DTD, но если такового не имеется, то процессор не сможет сообщить, является ли документ допустимым.

DTD оказывается особенно ценным, когда над рядом документов работает целая группа специалистов. В этом случае DTD служит общим справочником, где каждый разработчик может узнать, какие элементы являются допустимыми и как их можно применять.

Например, DTD для списка сотрудников может выглядеть следующим образом (листинг 19.7, *а*).

Листинг 19.7, *а*. Файл Emailbook.dtd

```
<!ELEMENT employee (person)>
<!ELEMENT person (lastname, firstname, email)>
<!ELEMENT lastname (#PCDATA)>
<!ELEMENT firstname (#PCDATA)>
<!ELEMENT email (#PCDATA)>
```

При наличии DTD в объявлении XML появляется атрибут `standalone="no"`, который информирует, что в документе теперь присутствуют определения DTD. Следующая же строка документа XML указывает тип документа. В объявлении типа документа указывается имя корневого элемента, в нашем случае `employee`. Кроме того, в нем указывается, какому определению типа документа соответствует данный XML-документ. В рассматриваемом случае, `Emailbook.dtd` (листинг 19.7, *б*).

Листинг 19.7, *б*. Файл Emailbook.xsl

```
<?xml version="1.0" standalone="no" ?>
<!DOCTYPE employee SYSTEM "emailbook.dtd">
<employee>
    <person>
        <lastname>Bond</lastname>
        <firstname>James</firstname>
        <email>bond007@yandex.com</email>
    </person>
    <person>
        <lastname>Cooper</lastname>
        <firstname>Gary</firstname>
        <email>gcooper@yandex.com</email>
     </person>
</employee>
```

Внутренний блок DTD

Объявление типа документа может указывать как на внешний блок (т. е. на DTD, хранящееся в отдельном документе, как это было в предыдущем примере), так и на внутренний блок (DTD, входящее в состав XML-документа). Следующий код (листинг 19.8) демонстрирует объявление типа документа, ссылающееся на внутренний блок.

Листинг 19.8. Внутреннее объявление

```
<?xml version="1.0"?>
<!DOCTYPE  animal[
   <!ELEMENT name (#PCDATA)>
   <!ELEMENT voice (#PCDATA)>
]>
<animal>
   <name> dog</name>
   <voice> gaf</voice>
</animal>
```

Элемент

Базовым типом объявлений DTD является объявление элемента `<!ELEMENT>`. В объявлении элемента указывается то, что может появиться в качестве его содержания.

```
<!ELEMENT имяЭлемента правило>
```

Если такового нет, то можно указать ключевое слово `EMPTY`. Например:

```
<!ELEMENT img EMPTY>
```

Задание частоты и альтернативы использования элемента

При определении элементов допустимо использование следующих символов:

- `?` — указывает на то, что элемент может быть использован один раз или не использован совсем;
- `*` — элемент может быть использован неограниченное число раз;
- `+` — элемент должен быть использован один или более раз.

Символ "`|`" позволяет указывать, что элемент появляется как альтернатива выбора из нескольких других элементов. Например:

```
<!ELEMENT remarkonbook(title, author+, remark*, output+|conclusion)>
```

Здесь элемент `remarkonbook` содержит один или более элементов `author`, может как содержать, так и не содержать любое число элементов `remark`, и

имеет либо один элемент conclusion, либо по крайней мере один элемент output.

Объявление атрибутов элемента

Атрибуты используются для связи пар "имя—значение" и применяются для того, чтобы сообщить дополнительную информацию об элементе. Списки атрибутов элемента определяются с помощью ключевого слова !ATTLIST. Внутри него задаются названия атрибутов, типы их значений и дополнительные параметры.

```
<!ATTLIST element name type defaultvalue ?>
```

Например,

```
<person department="Special" nickname="Agend007" sex="male">
   <firstname> James </ firstname>
   <lastname> Bond </ lastname >
</person>
```

Обратите внимание, что описать элемент <person> можно было за счет введения новых элементов вместо атрибутов.

```
<person>
   <firstname> James </ firstname>
   <lastname> Bond </ lastname >
   <department>Special</department>
   <nickname>Agend007</nickname>
   <sex>male</sex>
</person>
```

Приведем объявление атрибутов данного XML-документа.

```
<!ELEMENT person (firstname,lastname)>
<!ATTLIST person nickname CDATA #IMPLIED>
<!ATTLIST person department CDATA #REQUIRED>
<!ATTLIST person sex (male|female) "male">
<!ELEMENT employee (person)>
<!ELEMENT lastname (#PCDATA)>
<!ELEMENT firstname (#PCDATA)>
```

В этом примере для элемента person определяются три атрибута: nickname, department и sex, первые два из которых имеют тип CDATA, а последний — список возможных значений. Всего существует пять возможных типов значений атрибута:

- CDATA — содержимым документа могут быть любые символьные данные;
- ID — определяет уникальный идентификатор элемента в документе;

- IDREF (IDREFS) — указывает, что значением атрибута должно выступать название (или несколько таких названий, разделенных пробелами во втором случае) уникального идентификатора, определенного в этом документе элемента;
- ENTITY (ENTITIES) — значение атрибута должно быть названием (или списком названий, если используется ENTITIES) компонента (макроопределения), определенного в документе;
- NMTOKEN (NMTOKENS) — содержимым элемента может быть только одно отдельное слово (т. е. этот параметр является ограниченным вариантом CDATA);

Также в определении атрибута можно использовать следующие параметры:

- #REQUIRED — определяет обязательный атрибут, который должен быть задан во всех элементах данного типа;
- #IMPLIED — атрибут не является обязательным;
- #FIXED — указывает, что атрибут должен иметь только указанное значение, однако само определение атрибута не является обязательным, но в процессе разбора его значение в любом случае будет передано программе-анализатору.

Как сохранить набор записей в XML-формате

Начиная с ADO 2.5 появилась возможность сохранять набор записей в XML-формате. Этой операцией управляет метод Save объекта Recordset. Причем, достаточно в качестве значения его параметра PersistFormat указать константу adPersistXML. Следующий код (листинг 19.9) выводит таблицу tblPhone базы данных Phonebook.mdb в виде XML-файла Test.xml, а также в виде потока, который выводится в окно отладки.

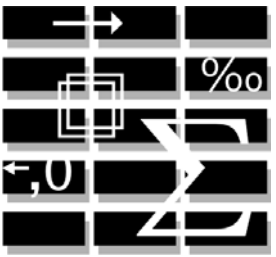
Листинг 19.9. Сохранение набора записей в виде XML-файла

```
Public Sub SaveToXML(rs As ADODB.Recordset)
   Dim adoStream As ADODB.stream
   On Error GoTo ErrorHandler
   Set adoStream = New ADODB.stream
   rs.Save Destination:=adoStream, PersistFormat:=adPersistXML
   rs.Save Destination:="test.xml", PersistFormat:=adPersistXML
   Set adoStream = Nothing
   Exit Function
```

```
ErrorHandler:
   MsgBox "Error"
End Sub

Public Sub DemoSaveToXML()
   Dim rs As New ADODB.Recordset
   rs.CursorType = adOpenKeyset
   rs.LockType = adLockOptimistic
   rs.Source = "SELECT * FROM Абонент ORDER BY Фамилия"
   rs.ActiveConnection = "DSN=ph"
   rs.Open
   SaveToXML(rs)
End Sub
```

ГЛАВА 20



Справочная система

Создать справочную систему можно при помощи программы HTML Help Workshop, бесплатно распространяемой корпорацией Microsoft и свободно доступной на ее сайте. Эта программа позволяет:

- конструировать справочную систему приложения на базе HTML-файлов;
- организовать HTML-файлы, образующие справочную систему, в один проект (HHP-файл);
- построить таблицу содержания системы (HHC-файл);
- создать ее индекс (HHK-файл).

Этапы построения справочной системы

Этапами построения справочной системы являются:

- создание разделов справочной системы;
- построение таблицы содержания;
- конструирование файла проекта;
- создание индекса.

Создание разделов справочной системы

Разделы справочной системы должны быть представлены в виде отдельных HTML-файлов. Поэтому разделы могут быть созданы в любом HTML-редакторе, например MS FrontPage, и даже просто в Блокноте. Разумно их создание и в MS Word с последующим сохранением в формате Web-страницы. В этом случае надо учитывать, что в HTML-файле, созданном в результате экспорта из DOC-файла, рисунки создаются не как теги `<img>`, а как вложенные OLE-объекты, что не всегда удобно.

Прежде чем создавать разделы, их предварительно надо структурировать. Например, создадим раздел **Об авторе** справочной системы, описывающей интегрированную среду разработки проекта (табл. 20.1).

Таблица 20.1. *Данные для справочной системы*

Раздел справочной системы	Файл справки
Об авторе	
Оглавление	content.htm
Справка	about.htm
Книги об офисном программировании	
Самоучитель VBA	book1.htm
Microsoft Excel 2000: Разработка приложений	book2.htm
Excel, VBA, Internet в экономике и финансах	book3.htm

Построение таблицы содержания

Для построения таблицы содержания (HHC-файла):

1. Запустите программу HTML Help Workshop.

   Выполните команду **File | New**. В отобразившемся окне **New** в списке **Specify what to create** выберите команду **Table of Contents**. Нажмите кнопку **OK**.

2. На экране отобразится окно создания таблицы содержания (рис. 20.1). В этом окне:

   - кнопка  отображает окно **Table of Contents Properties**, предназначенное для установки значений свойств таблицы содержания;
   - кнопка  вставляет заголовок раздела (heading) в таблицу содержания;
   - кнопка  вставляет страницу раздела (page) в таблицу содержания. Если у раздела таблицы содержания имеется подраздел, то надо вставлять заголовок, а если такового нет, то страницу раздела;
   - кнопка  позволяет редактировать выбранный элемент таблицы содержания;
   - кнопка  удаляет выбранный элемент таблицы содержания;
   - кнопки , , , позволяют изменять порядок разделов и их уровни.

3. Используя кнопки и и отображаемые при этом окна **Table of Contents Entry** (рис. 20.2), создайте таблицу содержания и свяжите ее

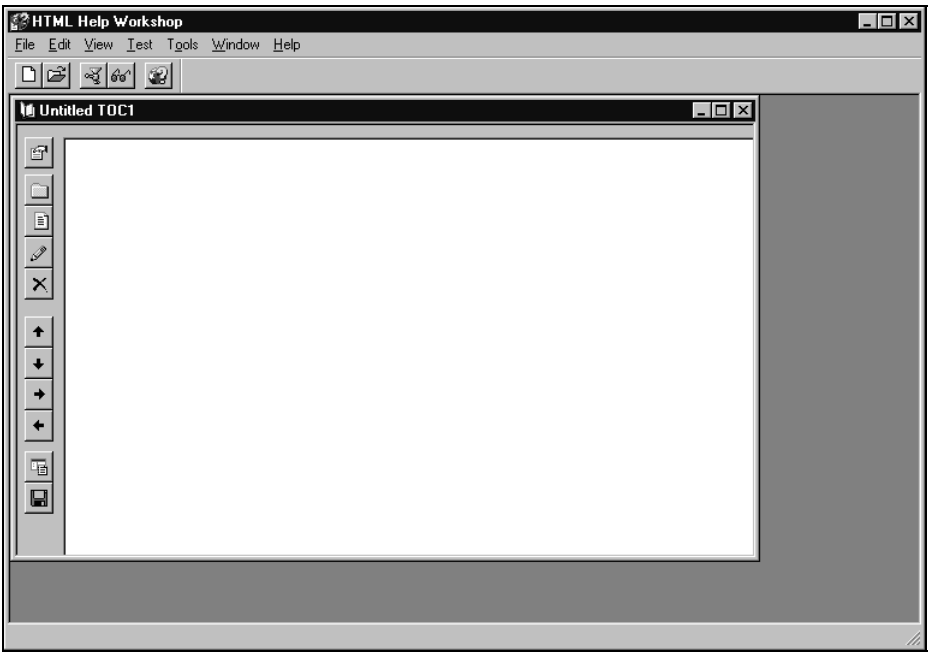


Рис. 20.1. Окно создания таблицы содержания

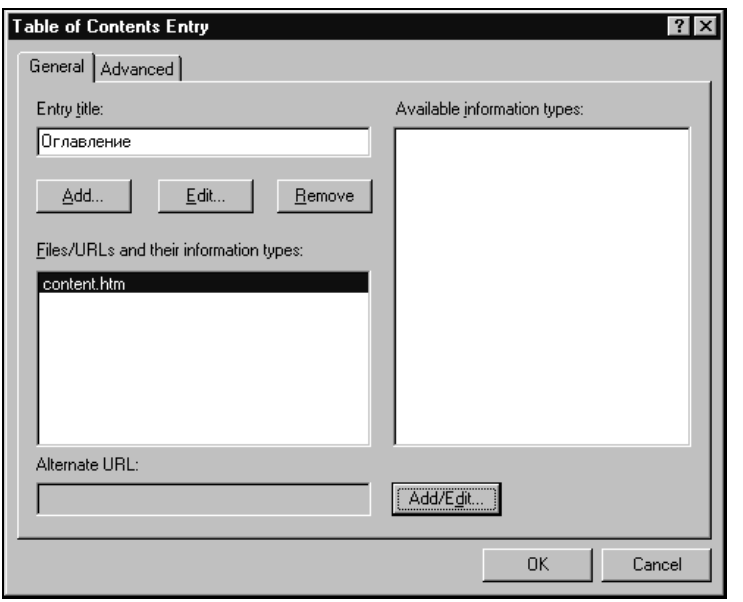


Рис. 20.2. Окна **Table of Contents Entry**

элементы, если это необходимо, с соответствующими HTML-файлами. На рис. 20.3 показана заполненная таблица содержания.

4. Сохраните файл, выбрав команду **File | Save File**.

Таблица содержания создана.

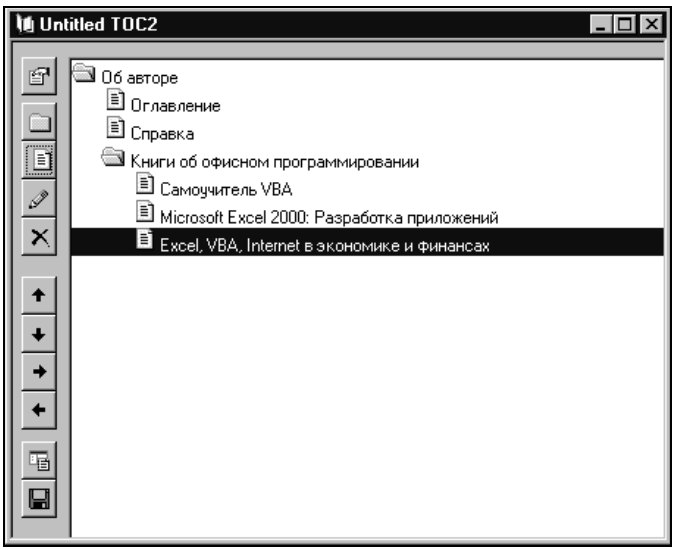


Рис. 20.3. Окно с созданной таблицей содержания

Конструирование файла проекта

Для построения файла проекта (HHP-файла):

1. Запустите программу HTML Help Workshop.
2. Выполните команду **File | New**. В отобразившемся окне **New** в списке **Specify what to create** выберите команду **Project**. Нажмите кнопку **OK**.
3. На экране отобразится первое окно мастера проекта (рис. 20.4). Нажмите кнопку **Next**.
4. На экране отобразится второе окно мастера проекта (рис. 20.5). В поле, используя кнопку **Browse**, введите полное имя проекта. Нажмите кнопку **Next**.
5. Появится третье окно мастера проекта (рис. 20.6). Флажок **HTML Help table of contents (.hhc)** устанавливается, если проект будет основан на уже существующем файле с таблицей содержания. В данном случае таковая имеется, поэтому установите этот флажок. Флажок **HTML Help index (.hhk)** определяет, используется ли в проекте индекс. В нашем случае

индекс не применяется, поэтому не устанавливайте этот флажок. Флажок **HTML files (.htm)** задает, применяются ли в проекте файлы разделов. В нашем случае установите этот флажок. Нажмите кнопку **Next**.



Рис. 20.4. Первое окно мастера проекта



Рис. 20.5. Второе окно мастера проекта

6. На экране отобразится четвертое окно мастера проекта (рис. 20.7). В поле, используя кнопку **Browse**, введите ссылку на файл с таблицей содержания. Нажмите кнопку **Next**.



Рис. 20.6. Третье окно мастера проекта

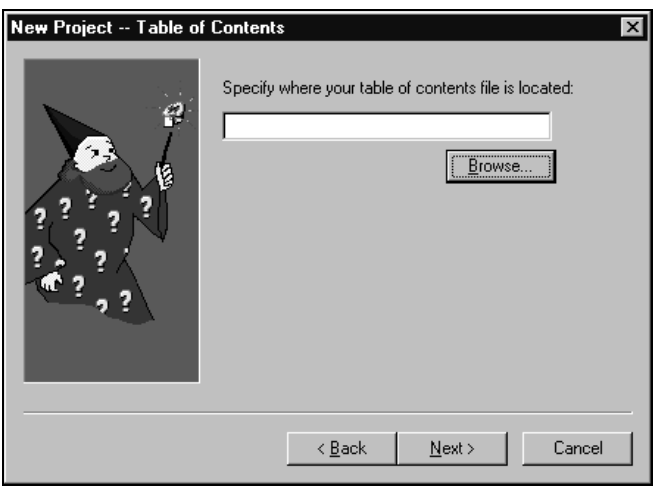


Рис. 20.7. Четвертое окно мастера проекта

7. На экране отобразится пятое окно мастера проекта (рис. 20.8). В список **Where your .htm files are located** при помощи кнопок **Add** и **Remove** добавьте или удалите файлы разделов проекта. Нажмите кнопку **Next**.
8. На экране отобразится шестое окно мастера проекта (рис. 20.9), информирующее о том, что все данные для конструирования проекта введены. Нажмите кнопку **Finish**.

Итак, HHP-файл проекта создан. По своей структуре он является текстовым файлом, который может, например, выглядеть так, как в листинге 20.1.



Рис. 20.8. Пятое окно мастера проекта



Рис. 20.9. Шестое окно мастера проекта

Листинг 20.1. Созданный файл проекта Author.hhp

```
[OPTIONS]
Contents file=F:\Vba2002\author.hhc

[FILES]
content.htm
about.htm
```

```
book1.htm
book2.htm
book3.htm
```

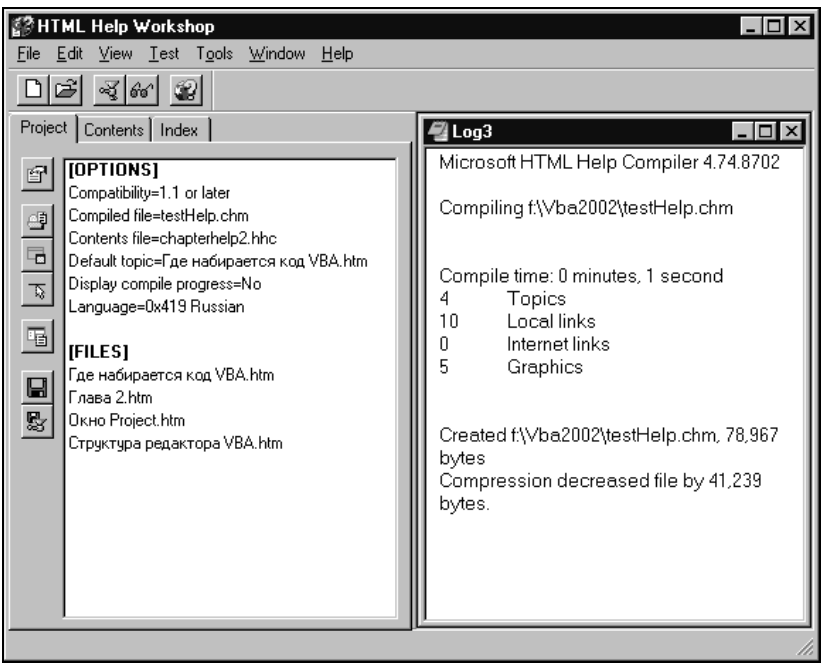


Рис. 20.10. Результат компиляции проекта

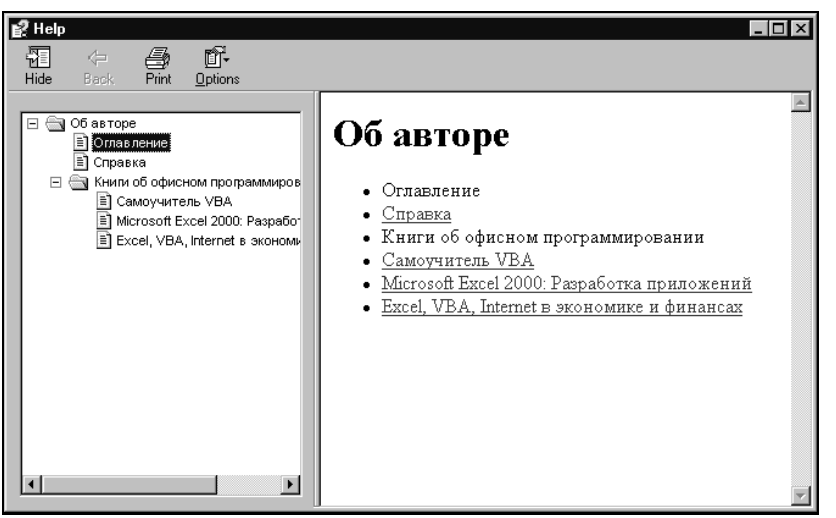


Рис. 20.11. Просмотр справочной системы

Теперь надо откомпилировать проект. Для этого выберите команду **File | Compile**. В окне HTML Help Workshop отобразится сообщение о результатах компилирования проекта (рис. 20.10). Откомпилированная справочная система будет храниться в CHM-файле.

Для просмотра созданной справочной системы выберите команду **View | Compiled File** (рис. 20.11).

Создание индекса

Индекс позволяет производить ускоренный поиск информации в справочной системе по ключевым словам. В HTML Help Workshop он конструируется элементарно. Для этого достаточно выбрать вкладку **Index** окна проекта HTML Help Workshop, а затем выполнить очевидные действия, предлагаемые мастером создания индекса по заданию ключевых слов и ассоциированных с ними HTML-файлов разделов.

Подключение справки

Наиболее простым способом подключения справки является метод `Help` объекта `Application`. В качестве значения его параметра `HelpFile` достаточно указать полное имя соответствующего CHM-файла. Например, в следующем коде (листинг 20.2) при нажатии на кнопку отображается справка Author.chm, созданная в предыдущем разделе.

Листинг 20.2. Подключение справки при помощи метода `Help`

```
Private Sub cmdHelp_Click()
   Application.Help HelpFile:=ThisWorkbook.Path & "\author.chm"
End Sub
```

Справку можно также вызвать оператором `Shell`, запускающим программу hh.exe, которая и открывает справку на экране (листинг 20.3).

Листинг 20.3. Подключение справки при помощи оператора `Shell`

```
Private Sub cmdHelp_Click()
   Call Shell("hh.exe " & ThisWorkbook.Path & "\author.chm", _
              vbNormalFocus)
End Sub
```

Создание контекстно-зависимой справки

Для создания контекстно-зависимой справки откройте HHP-файл проекта в Блокноте. Добавьте в него два новых раздела: `[ALIAS]` и `[MAP]`. Раздел `[ALIAS]` определяет идентификаторы для идентификации разделов справки, а раздел `[MAP]` ассоциирует с этими идентификаторами числовые значения, которые и будут передаваться через параметр `HelpContextID` метода `Help` для вызова соответствующего раздела справки. Сохраните измененный HHP-файл проекта (листинг 20.4) и откомпилируйте его заново.

Листинг 20.4. Файл проекта Author.hhp

```
[ALIAS]
IDH_content=content.htm
IDH_about=about.htm
IDH_book1=book1.htm
IDH_book2=book2.htm
IDH_book3=book3.htm

[MAP]
#define IDH_content 1
#define IDH_about 2
#define IDH_book1 11
#define IDH_book2 12
#define IDH_book3 13
```

Подключение контекстной справки

Наиболее простым способом подключения контекстной справки является вызов метода `Help` объекта `Application` с двумя параметрами. Первый из них указывает полное имя файла, а второй — числовое значение соответствующего раздела справки. Например, в следующем коде (листинг 20.5) при нажатии первой кнопки отображается раздел **Справка**, а при нажатии второй кнопки — раздел **Самоучитель VBA** созданной нами справочной системы Author.chm.

Листинг 20.5. Подключение справки при помощи оператора `Shell`

```
Private Sub cmdAbout _Click()
   Application.Help HelpFile:=ThisWorkbook.Path & "\author.chm", _
                    HelpContextID:=2
End Sub
```

```
Private Sub cmdBook1_Click()
   Application.Help HelpFile:=ThisWorkbook.Path & "\author.chm", _
                    HelpContextID:=11
End Sub
```

Подключение контекстно-зависимой справки с использованием Помощника Microsoft Office

В MS Office имеется очень удобное средство — Помощник, которое максимально приспособлено для отображения справочной информации. Рассмотрим простой пример его использования (листинг 20.6), причем Помощник будет отображаться при щелчке правой кнопкой мыши на рабочем листе (рис. 20.12). Конечно, для реализации этого проекта файлы справочной системы должны находиться в каталоге с данной рабочей книгой.

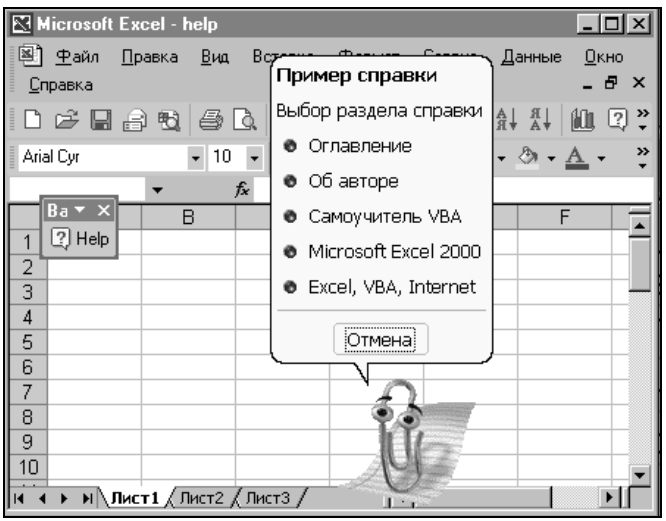


Рис. 20.12. Подключение контекстно-зависимой справки с использованием Помощника Microsoft Office

Листинг 20.6. Отображение справки с использованием Помощника Microsoft Office

```
Private Sub Worksheet_BeforeRightClick(ByVal Target As Range, _
      Cancel As Boolean)
   Cancel = True
   ShowAssistent
End Sub
```

```
Private Sub ShowAssistent()
   Dim choice As Integer
   If Not Assistant.Visible Then
      With Assistant
         .Visible = True
         .Animation = msoAnimationThinking
      End With
   Else
      Assistant.Animation = msoAnimationThinking
   End If

   With Assistant.NewBalloon
      .BalloonType = msoBalloonTypeButtons
      .Heading = "Пример справки"
      .Text = "Выбор раздела справки"
      .Labels(1).Text = "Оглавление"
      .Labels(2).Text = "Об авторе"
      .Labels(3).Text = "Самоучитель VBA"
      .Labels(4).Text = "Microsoft Excel 2000"
      .Labels(5).Text = "Excel, VBA, Internet"
      .Button = msoButtonSetCancel
      .Mode = msoModeModal
      choice = .Show
   End With

   Dim helpFile As String
   helpFile = ThisWorkbook.Path & "\author.chm"

   Select Case choice
      Case 1
         Application.Help helpFile, 2
      Case 2
         Application.Help helpFile, 1
      Case 3
         Application.Help helpFile, 11
      Case 4
         Application.Help helpFile, 12
      Case 4
         Application.Help helpFile, 13
   End Select
End Sub
```

Примечание

У Помощника имеется свой метод `Help`. Этот метод отображает не контекстную справку, а встроенное окно Помощника **What would you like to do?** для справки по Microsoft Office.

Подключение контекстно-зависимой справки при помощи панели инструментов

При создании приложений бывает удобно подсоединить вызов справки к соответствующей кнопке панели инструментов (см. рис. 20.12). Процедура `ToolHelp` из приводимого ниже проекта (листинг 20.7) как раз и демонстрирует, как это делается. Данная процедура создает панель инструментов, в которой располагается единственная кнопка  (ее `ID` равен `984`). Нажатие же кнопки вызовет отображение на экране справки.

Листинг 20.7, *а*. Отображение справки из панели инструментов. Модуль ЭтаКнига

```
Private Sub Workbook_Open()
   ToolHelp
End Sub

Private Sub ToolHelp()
   Dim cmdBar As CommandBar
   Dim btnHelp As CommandBarButton
   Set cmdBar = Application.CommandBars.Add(Name:="BarWithHelp", _
                                            Temporary:=True)
   Set btnHelp = cmdBar.Controls.Add(Type:=msoControlButton, _
                                     ID:=1, _
                                     Temporary:=True)
    With btnHelp
       .Style = msoButtonIconAndCaption
       .Caption = "Help"
       .OnAction = "CallHelp"
       .FaceId = 984
    End With
    cmdBar.Visible = True
End Sub

Private Sub Workbook_BeforeClose(Cancel As Boolean)
   Application.CommandBars("BarWithHelp").Delete
End Sub
```

Листинг 20.7, *б*. Отображение справки из панели инструментов. Модуль ЭтаКнига

```
Sub CallHelp()
   Application.Help ThisWorkbook.Path & "\author.chm"
End Sub
```