Усовершенствуйте приложения Microsoft Office, WordPerfect Office, AutoCAD, Visio и многие другие!

VBA

для "чайников"™

-е издание

Простой способ усовершенствования бизнес-приложений с помощью Visual Basic for Applications

Для сомневающихся

Стив Каммингс

автор книги *Microsoft Office 2000 Secrets*

ДИАЛЕКТИКА

VBA для "ЧАЙНИКОВ"™

3-е издание

VBA FOR DUMMIES®

3RD EDITION

by Steve Cummings

ЭЛЕКТРОННАЯ БИБЛИОТЕКА
Прочитал? Удали!
LIBRUS.RU

Hungry Minds™
HUNGRY MINDS, INC.
Best-Selling Books • Digital Downloads • e-Books • Answer Networks • e-Newsletters • Massive Web Sites • e-Learning
New York, NY • Cleveland, OH • Indianapolis, IN

VBA для "ЧАЙНИКОВ"™

3-е издание

Стив Каммингс

ДИАЛЕКТИКА

Москва • Санкт-Петербург • Киев
2002

ББК 32.973.26-018.2.75
К18
УДК 681.3.07

Компьютерное издательство "Диалектика"
Зав. редакцией *В. В. Александров*

Перевод с английского и редакция *КБ. Тараброва*

По общим вопросам обращайтесь в издательство "Диалектика"
по адресу: info@dialektika.com, http://www.dialektika.com

Камминг,Стив.

К18 VBA для "чайников", 3-е издание. : Пер. с англ. — М. : Издательский дом "Вильямс", 2001. — 448 с. : ил. — Парал. тит. англ.

ISBN 5-8459-0230-4 (рус.)

Эта книга поможет освоить интегрированную среду разработки VBA и научиться программировать в VBA с использованием объектов, их свойств, методов и событий. Обсуждаемые приемы программирования иллюстрируются примерами, которые можно сразу же опробовать на практике. Полученные знания вы сможете применить как для настройки и усовершенствования популярных офисных приложений, включая приложения Office XP, так и для создания собственных приложений.

Книга рассчитана на тех, кто собирается быстро и без лишних усилий научиться программировать для Windows, используя VBA.

ББК 32.973.26-018.2.75

Все названия программных продуктов являются зарегистрированными торговыми марками соответствующих фирм.

Никакая часть настоящего издания ни в каких целях не может быть воспроизведена в какой бы то нн было форме и какими бы то ни было средствами, будь то электронные или механические, включая фотокопирование и запись на магнитный носитель, если на это нет письменного разрешения издательства Hungry Minds, Inc.

Copyright © 2002 by Dialektika Computer Publishing.
Original linglish language edition copyright © 2001 by Hungry Minds. Inc.

All rights reserved including the right of reproduction in whole or in part in any form.
This edition published by arrangement with the original publisher. Hungry Minds. Inc.

For Dummies and Dummies Man are trademarks under exclusive license to Hungry Minds. inc. Used by permission.

ISBN 5-8459-0230-4 (рус.)
ISBN 0-7645-0856-3 (англ.)

© Компьютерное издательство "Диалектика". 2002
© Hungry Minds. Inc., 2001

Оглавление

Содержание

Об авторе

Стив Кяммингс занимается программированием больше 20 лет, используя такие разные языки программирования, как ассемблер, COBOL и C++, а также VBA и Visual Basic. Он автор и соавтор более десятка компьютерных книг, среди которых и книга *Секреты Office 97,* выпушенная издательством "Диалектика". Кроме того, ему принадлежат сотни статей, опубликованных в известных компьютерных журналах, в том числе *PC World, Macworld, PC Magazine, PC/Computing* и *PC Week*.

Посвящение

Моей бабушке по поводу 97-летия

Благодарности

Спасибо всем сотрудникам издательства *Hungry Minds,* а особенно моим редакторам: Джеймсу Расселу (James Russel), который проявил немало внимания и терпения при работе над настоящей книгой, а также Джейд Вильямс (Jade Williams) и Кэлли Оливер (Kelly Oliver). Я благодарен компании *V Communications* (www. v-com. com) за предоставление таких программных продуктов, как System Commander и Partition Commander. Это замечательные утилиты для настройки работы нескольких операционных систем на одном компьютере, что мне требовалось при работе с beta-версиями Microsoft Office XP. Также спасибо Лизе Роббинс (Lisa Robbins) из компании *Waggener Edstrom,* подразделения компании *Microsoft,* за помощь при знакомстве с нюансами новой версии VBA, а также за предоставление пробных версий программных продуктов компании *Microsoft,* необходимых мне при работе.

Введение

Да, это именно та книга, которая необходима для того, чтобы начать освоение VBA (аббревиатура от Visual Basic for Applications, что означает Visual Basic для приложений). Благодаря этой книге вы узнаете об основных принципах программирования в VBA и получите необходимые навыки для создания полезных программ. А самое главное, вы сможете сделать это без лишних усилий.

В книге использовано много примеров, написана она легко и понятно, поскольку, кто знает, сколько скучных томов уже издано в этом мире! *Я* попытался изложить все самым обычным языком, по возможности исключив из употребления режущий ухо нормального человека специальный жаргон. И с этого момента я начинаю отпускать (иногда не самые лучшие) шуточки, чтобы вам было на что направить свое раздражение.

С другой стороны, обсуждение рассматриваемых вопросов не слишком упрощено, иначе оно было бы лишено всякой ценности. Если отбросить шутки в сторону, то эта книга — полноценное справочное пособие, охватывающее все основные разделы VBA.

О чем эта книга

Вы, наверное, уже знаете, что VBA — это язык программирования, встроенный во множество программ, от приложений Microsoft Office, Microsoft Project, Visio и AutoCAD до многочисленных специализированных приложений, предназначенных для управления производственными процессами, учета финансовых ресурсов или информационной поддержки клиентов.

В этой книге рассмотрены все существенные аспекты программирования в VBA. Здесь вы найдете достаточно полную информацию по следующим темам:

- ✓ использование преимуществ визуальных средств программирования VBA;
- ✓ запись и редактирование макросов;
- ✓ запуск VBA-программ из других приложений;
- ✓ создание приятных на вид диалоговых окон и других элементов интерфейса;
- ✓ работа с объектами, которая станет ключом к использованию всей силы VBA-приложений.

В настоящей главе рассмотрены версии VBA с версии 6 по 6.3 (VBA 6.3 входит в состав приложений Microsoft Office XP). Согласно данным компании *Microsoft,* все версии VBA, начиная с версии 6 и заканчивая версией 6.3, с точки зрения программиста совершенно идентичны. Компания *Microsoft* исправила некоторые ошибки, улучшила производительность, но способы написания программного кода или создания диалоговых окон не изменились. Поэтому, когда я говорю о VBA 6, это касается как VBA 6.0 и VBA 6.3, так и всех промежуточных версий.

Неследовало бы делать предположений...

Но я все же сделаю. *Я* предполагаю, что вы не такой уж и "чайник", наоборот, думаю, вы должны весьма комфортабельно чувствовать себя в Windows. Поэтому, если вы не знаете, как обращаться с мышью, выбирать из меню и щелкать на кнопках, появляющихся на экране ва-

шего монитора, вам имеет смысл предварительно прочитать одну из книг, выпущенных издательством "Диалектика", например *Windows для "чайников"* (есть также отдельные издания по Windows 95, Windows 98, Windows Me и Windows 2000). Далее, чтобы использовать VBA, вы должны иметь по крайней мере одно приложение, в которое встроены средства разработки VBA-программ. Среди таких приложений прежде всего следует назвать лидера рынка программ для бизнеса Microsoft Office, за которым следует постоянно растущая группа продуктов, принадлежащих другим производителям. Изо всех многочисленных возможностей подойдет, например, любое из следующих приложений:

- ✓ любое приложение Microsoft Office — Word, Excel, PowerPoint, Access, Outlook или FrontPage;
- ✓ Microsoft Project;
- ✓ CorelDraw версии 9 или 10 и Corel WordPerfect Office 2000;
- ✓ серия графических бизнес-приложений iGrafx от Micrografx;
- ✓ Visio версий 4.5, 5 или 2000;
- ✓ AutoCAD R14, AutoCAD 2000 или AutoCAD 2000i для Windows;
- ✓ Autodesk Map;
- ✓ TurboCAD Professional;
- ✓ M.Y.O.B. Accounting Software;
- ✓ пакет программ для учета ресурсов Great Plains (некоторые программы из этого пакета понимают VBA);
- ✓ OmniTrader, средства оценки рисков и управления торговыми операциями.

Разделяй и властвуй

Теоретические знания и навыки, которые требуются для программирования в VBA, образуют нечто целое, органический симбиоз взаимосвязей, подобный самой жизни... Именно по этой причине мне пришлось разбить предлагаемый материал на пять больших частей, каждая из которых содержит как минимум две главы.

Часть I. "Начальные сведения о VBA"

Начинает книгу только одна унылая глава, посвященная теоретическим вопросам, зато следующие три шустренькие главы разбудят вас и заставят бежать вместе с VBA. Вы научитесь записывать макросы, чтобы не обращаться к программированию там, где это можно и оправданно, запускать свои VBA-программы из других приложений. Вам даже предстоит по ходу дела создать полностью законченную программу, выполняющую определенное практическое задание.

Часть II. "Курс программирования на VBA"

Редактор Visual Basic является, так сказать, доверенным представителем VBA на экране — здесь вы пишете свои программы, конструируете открываемые этими программами окна и тестируете свои создания, чтобы выяснить причины, по которым они отказываются работать так, как нужно. Об этом я расскажу в первой главе части. Затем я расскажу об использовании переменных. В остальных главах обсуждаются способы контроля того, что происходит при выполнении программы, а также способы выявления и исправления хотя бы

части тех ошибок, которые всегда норовят прицепиться к любой программе. Завершим мы материал части II рассмотрением таких вопросов, как создание окон, диалоговых окон и форм, а также приемов программирования, необходимых для этого.

Часть III. "Практикуемся в программировании на VBA"

Эта часть — сердце книги. Здесь, в сотворенных с любовью главах, я открою вам секреты мира VBA. Первая половина глав этой части систематизирует компоненты VBA-программы и раскрывает структуру этих компонентов, чтобы вы всегда точно знали, что и где вам следует печатать. Далее идет практикум по правильному присвоению имен в VBA и приданию презентабельного вида программному коду. Затем следуют главы, рассказывающие о работе с переменными и объектами, что очень важно при настройке таких приложений, как Word, Excel или CorelDraw. В последней главе мы поговорим о обработке данных, представленных в виде массивов и наборов.

Часть IV. "Профессиональная работа с VBA"

В главах этой части вы познакомитесь с приемами программирования, которые применимы практически ко всем приложениям Office, например настройка интерфейса пользователя, программирование помощника по Office и многое другое. Затем мы подробно поговорим о программировании для Word и Excel, а также вопросах, представляющих немалый интерес для VBA-программистов, независимо от того, с какими приложениями они работают.

Часть V. "Великолепные десятки"

Ваше путешествие в страну программирования средствами VBA завершается двумя главами, посвященными самым разнообразным темам. Сначала мы обсудим более изысканные приемы программирования. Конечно, обсуждение не будет слишком глубоким, но и поверхностным его тоже нельзя назвать — вполне достаточно для того, чтобы вы смогли реально использовать эти приемы. Далее приводится каталог доступных ресурсов VBA, охватывающий как информационные ресурсы, так и программные продукты. Туда стоит заглянуть и для того, чтобы пополнить свои знания в VBA-программирования, и для того, чтобы пополнить свою библиотеку программных средств разработчика.

Как использовать примеры

Эта книга содержит достаточно много примеров программного кода, призванного иллюстрировать те концепции, которые я пытаюсь объяснить. Примеры кода будут ясно выделены моноширинным шрифтом, причем и в случае представления программного кода отдельным блоком, как здесь:

```
WhatEverItIs.Color = "Chartreuse"
```

и в случае появления программного кода просто в строке, как здесь: `Debug.Print`.

Для экономии места эти примеры чаще всего будут лишь фрагментами, которые сами по себе выполняться не способны. Такие фрагменты выполнимы только в составе определенных процедур, а многие примеры как раз и не включают необходимых операторов определения этих процедур. В общем, если вы пожелаете увидеть результат выполнения этих примеров на своем собственном компьютере, вам придется заключить представленные в примерах операторы в подходящую процедуру (а в некоторых случаях дополнить программный код в соответствии с приведенными в тексте объяснениями).

Лично я считаю, что процесс печатания программного кода укрепляет моральные устои, но я признаю также, что на некоторых из вас такой аргумент не произведет должного впечатления. Именно по этой причине тексты процедур, соответствующих примерам, в готовом виде размещены на сервере издательства "Диалектика" и теперь доступны через Internet по адресу www.dialektika.com. Так что не пугайтесь, печатать тексты процедур заново вам не придется — вы можете выгрузить их на свой компьютер и без лишних усилий просто импортировать нужную процедуру в открытый проект, сразу получив возможность ее выполнения.

Довольно часто VBA-операторы (отдельные единицы программного кода) оказьшаются достаточно длинными и поэтому случается, что они не умещаются в одной строке книги. В таких случаях для обозначения того места, где единый оператор переносится на новую строку, я использую стандартно применяемый для этого в VBA символ подчеркивания (_).

Также я использовал в настоящей книге следующие соглашения.

- ✓ Если я хочу, чтобы воспользовались определенной командой из меню, я говорю вам "выберите команду Файл⇨Открыть". Это означает, что вы должны подвести указатель мыши к меню Файл и выбрать из него команду Открыть.
- ✓ Новые элементы выделяются *курсивом.*
- ✓ Части команд, которые вам следует изменить, выделяются *так.*
- ✓ Команды, которые вам необходимо ввести, отображаются полужирным.
- ✓ Иногда полужирное выделение используется в инструкциях.

Пиктограммы, используемые в данной книге

Вы, наверное, видели уже немало книг, на полях которых встречаются пиктограммы, призванные помочь быстро найти самую важную информацию. Для этой книги изо всех пиктограмм я выбрал только четыре, чтобы не создавать внутри книги отдельное руководство по использованию пиктограмм.

Обозначает любую информацию, на которую я счел нужным обратить ваше внимание.

Материал, отмеченный этой пиктограммой, ненамного труднее для понимания, чем остальной материал книги.

Предупреждает о возможной опасности.

Так я обозначил примеры с достаточно большими текстами программного кода, чтобы вы не сомневались, что вам нет необходимости набирать код вручную, если потребуется его использовать. Если же возле текста, представляющего программный код, такой пиктограммы нет, все равно не отчаивайтесь— тексты программного кода многих не слишком длинных примеров есть также на сервере издательства "Диалектика", но мне просто не хотелось, чтобы книга была напичкана пиктограммами сверх всякой меры.

Web-страница VBA для "чайников"

В дополнение к этой книге я планирую создание Web-страницы, адрес которой должен быть следующим:

`www.seldenhouse.com/vba`

Там вы найдете:

- ✓ дополнения и исправления к тексту книги;
- ✓ дополнительные материалы, по каким-либо причинам не вошедшие в книгу;
- ✓ исходные программные коды, доступные по мере того, как я буду получать их в свое распоряжение;
- ✓ ссылки на другие страницы, посвященные VBA, включая страницы, относящиеся к различным конкретным VBA-приложениям.

Но не ожидайте увидеть там чудеса оформления и множество графических изображений развлекательного характера — я предполагаю направить главные усилия на то, чтобы предоставленная информация оказалась максимально полезной. Поэтому, если вы обнаружите какие-то интересные приемы программирования или решения, о которых полезно знать другим людям, или если у вас есть ссылки на другие Web-страницы, я с благодарностью приму от вас сообщения об этом.

Часть I

Начальные сведения о VBA

"ФИЛИПП, привет! Mbt уже почти вошли в систему, но подскажи,
как подключиться непосредственно к экрану.
Я не хочу прославиться на весь город, как самый неудачливый хакер!"

В этой части...

В главе 1 обсуждаются те понятия, на которых основано программирование в VBA. Обсуждаемые здесь идеи действительно важны — поверьте мне, до тех пор пока вы не поймете их, серьезная работа с VBA будет как минимум трудной. Но как только вы их одолеете, эта книга станет для вас жутко полезной.

В главах 2–5 обсуждаются запись макросов, создание VBA-кода и запуск программ, включая те же макросы. После знакомства с этой частью вы уже будете точно знать, как оживить свои VBA-программы в приложениях.

Правда, вы **по-прежнему** *не будете знать, что же следует поместить в сами программы, но именно поэтому в данной книге* **есть** *еще около двух десятков глав.*

Глава 1

Уважайте теорию, теория —это все!

В этой главе...

- Для чего VBA-программы подходят лучше всего
- Как VBA-программы взаимодействуют с другими приложениями
- О цикле создания программы — планирование, разработка программного кода, создание форм и тестирование результата
- Почему VBA не настолько стандартизирован, как можно было бы предположить

Современные приложения, ориентированные на профессиональное использование, предлагают множество интересных, умных и мощных возможностей. Но даже в случае самых изощренных приложений довольно велика вероятность найти в них что-то, что безусловно требует усовершенствования по вашему мнению.

Вот тут-то и приходит на помощь VBA. Если ваше программное обеспечение слишком зажато с точки зрения дизайна, VBA поможет ослабить ремень ровно настолько, чтобы нужное приложение обеспечивало вам полный комфорт, будучи скроенным в соответствии с именно вашими кривыми и выпуклостями.

Зачем мучиться с VBA?

Не следует хвататься за VBA только потому, что эта аббревиатура то тут, то там впечатляюще произносится во время служебных вечеринок, Вам следует ближе познакомиться с VBA, если только вы хотите лучше приспособить свое программное обеспечение к тем запросам, которые выдвигаете вы или другие пользователи.

Вообще-то компьютер остается бесполезной вещью до тех пор, пока он не делает то, что нужно *вам*. Visual Basic для приложений — именно это и означает аббревиатура VBA — представляет собой набор средств программирования для создания ваших собственных программ. Как можно догадаться из самого названия, VBA предназначается для подгонки имеющихся приложений под запросы пользователя. Здесь должно быть ясно, что *приложение* — это полномасштабная программа, выполняющая серьезную практическую работу (например, текстовый процессор или приложение баз данных), в отличие от *утилиты,* которая призвана сделать вашу компьютерную систему более удобной и заставить работать ее надежнее.

С помощью VBA вы можете по своему усмотрению изменить внешний вид или способ применения имеющихся средств приложения и даже добавить свои, совершенно новые возможности. Вот несколько примеров, где VBA окажется наилучшим выбором.

✓ Возможность, которую вам приходится часто использовать, оказывается спрятанной где-то в глубинах диалоговых окон. Почему бы тогда не поместить соответствующую команду в виде кнопки на панель инструментов, чтобы вызывать команду одним щелчком? А еще лучше, если бы и сама эта кнопка появлялась только тогда, когда команда действительно может понадобиться!

- ✓ Вам часто приходится повторять один и тот же набор команд. Не предпочтете ли вы иметь программу, которая запомнит все необходимые для этого шаги и позволит запускать сразу всю последовательность одной командой?
- ✓ Ваше программное обеспечение просто не имеет пары-тройки возможностей, которые вам совершенно необходимы. Почему бы вам не добавить недостающие команды самостоятельно?

VBA позволяет выполнить любую из вышеприведенных модификаций приложения; более того, с помощью VBA вы можете создавать свои законченные и полностью работоспособные программы. Ниже, в разделе "Подробнее о том, что можно сделать в VBA", эти возможности обсуждаются более детально.

Хотите еще один аргумент в пользу того, что вам стоит-таки купить билет на VBA-экспресс? Пожалуйста: VBA быстро и неуклонно движется по направлению к тому, чтобы стать стандартом в индустрии создания программ. После освоения VBA вы сможете использовать этот язык в любом из приложений, поддерживающих VBA. (Правда, для каждого из VBA-приложений, с которыми вы будете работать, вам придется освоить также и их специальные VBA-жаргоны. Подробнее об этом — ниже в разделе "VBA как стандарт (точнее, род стандарта)".)

И если вы знаете VBA, вы автоматически превращаетесь в эксперта по Visual Basic. Еще одинпродукт фирмы *Microsoft,* Visual Basic представляет собой одно из наиболее популярных в мире средств разработки программ самого разного уровня — от простейших условно-бесплатных утилит до приложений высшего класса.

Кстати, одно из преимуществ VBA-стандарта объясняется тем, что вам нужно установить всего лишь одну копию тех файлов поддержки, которые потребуются при работе с VBA в любом из приложений. При стандартной установке все DLL-файлы (библиотеки динамической компоновки), как и другие необходимые для VBA файлы, размещаются в папке `\Program Files\Common Files\Microsoft SharedWBA`.

VBA — это штурвал, но к нему нужен еще и корабль

Теперь ясно, что VBA — прекрасное средство для усовершенствования приложений. Однако тут скрыта одна загвоздка: поддержка VBA должна быть встроенной в то приложение, которое вы собираетесь модифицировать. (В этой книге я буду использовать термин *VBA-приложение* для обозначения тех приложений, в которые встроена поддержка VBA, а те программы, которые создаются с помощью VBA, я буду называть *VBA-программами.*)

Microsoft — вы, наверное, уже слышали об этой компании — создала VBA и обеспечила поддержку VBA во всех своих главных приложениях: Office 97, Office 2000, Office XP, Word, Excel, Access и PowerPoint. В Outlook, единственном не упомянутом в этом ряду приложении из пакета Microsoft Office, для создания и изменения функциональных возможностей форм вы можете воспользоваться упрощенной версией VBA, получившей название VBScript. В отличие от вышедших раньше версий, Outlook 2002 (как и ее предшественница Outlook 2000) содержит полноценную поддержку VBA для модификации функциональных возможностей самой программы Outlook. VBA поддерживают

также FrontPage, средство автоматизации разработки Web-страниц и управления Web-узлами фирмы *Microsoft,* Visio — популярный пакет создания графики для бизнеса, а также Microsoft Project.

VBA-приложения третьих фирм

Многие разработчики программных средств получили лицензию фирмы *Microsoft* на использование технологии VBA. Вот список некоторых из доступных на сегодня VBA-приложений, предлагаемых третьими фирмами.

- ✓ **Corel WordPerfect Ofilce 2000,** Соперник Microsoft Office, включает текстовый процессор, приложения для обработки электронных таблиц, баз данных и для создания презентаций.
- ✓ **CorelDraw.** Являясь самым популярным пакетом для работы с рисунками, CorelDraw вместе с набором других программ из пакета позволяет создавать и обрабатывать как векторную графику, так и растровые изображения, печатать их либо размещать на Web-страницах.
- ✓ **AutoCAD.** Пользующийся большим успехом продукт фирмы *Computer Aided Design;* некоторые другие приложения этой фирмы также являются VBA-приложениями.
- ✓ **M.Y.O.B. Accounting.** Полный набор средств бухгалтерского учета для малого бизнеса.
- ✓ **Micrografx iGrafx.** Приложения для создания бизнес- графики — блок-схем, организационных диаграмм и т.п., а также для графического моделирования бизнес-операций.
- ✓ **OmniTrader.** Приложение для оценки и анализа рисков.

Встречаются приложения, в которые встроены средства программирования, очень похожие на VBA, но отличные от VBA. При этом, хотя вы и получаете возможность модифицировать эти приложения практически так же, как VBA-приложения, ваши VBA-программы выполняться в этих Не-VBA-приложениях не будут.

О распространенности VBA-приложений

Практически всегда можно утверждать, что приложение, которое легко модифицировать, чтобы оно отвечало нужным требованиям, обладает большей привлекательностью. По этой причине VBA встречается в приложениях почти любого вида — от самых популярных до самых узкоспециализированных.

В мире используются многие миллионы копий Microsoft Office, так что уже одни только приложения Office сами по себе окажутся вполне широким полем действия. Имеющие довольно широкую известность, VBA-приложения других фирм я уже тоже называл (такие как WordPerfect, Visio, AutoCAD).

Вместе с тем, многие VBA-приложения направлены на более узкие специализированные рынки. В качестве примера можно назвать приложения для управления производством, использующим роботизированные линии. Еще одним примером являются приложения для информационной поддержки, которые используются в группах технической поддержки, призванных помочь потребителям или персоналу других подразделений решать возникающие проблемы. VBA можно также обнаружить и в программном обеспечении для торговли, строительной инженерии, телефонии, обработки данных, управления потоками документов, финансового обслуживания, юридической поддержки и медицины.

Лицензии на VBA получили также целый ряд корпораций, которые не являются участниками рынка программного обеспечения. Эти компании занимаются разработкой приложений

с нуля для своих собственных нужд, учитывая специфические потребности каждой конкретной фирмы. И даже в этом случае они встраивают в свои приложения поддержку VBA, чтобы впоследствии эти приложения можно было модифицировать и расширять, затрачивая при •ЛЮМ минимум усилий.

Программирование на VBA

По сути, программировать— это значит говорить компьютеру, что он должен делать. Создавая компьютерную программу, вы даете компьютеру некоторый набор шагов, по которым ему предписано следовать. Например, вы можете указать компьютеру сделать следующее.

- ✓ Открыть окно, полное интересных кнопочек и меню.
- ✓ Создать поле и ввести в него дату вашего рождения.
- ✓ Щелкнуть на кнопке Вычислить возраст, которая вычислит ваш возраст, основываясь на введенной дате.

Еще не так давно создание программы означало необходимость самому выписывать все инструкции программы. Поскольку компьютеры не понимают человеческого языка, программистам нужно было печатать *программный код* — список всех таких инструкций, —- используя терминологию одного из специальных языков программирования.

VBA исключает необходимость печатать программный код, определяющий то, как ваша программа будет выглядеть на экране. Тем не менее написание программного кода вручную все еще требуется для любой программы, независимо от ее сложности. *Частью* VBA является язык программирования. Вот пример VBA-кода:

```
Sub SelectNextQuestion()
    If SelectionStrategy = Randomly Then
        AskRandomQuestion
    Else If CurrentQuestion = TotalQuestions Then
        CurrentQuestion = 1
    Else
        CurrentQuestion = CurrentQuestion +1
    End If
End Sub
```

Если вы немного знаете английский язык и прочли этот пример программного кода достаточно внимательно, вы должны догадаться, что здесь предполагается выполнить. Одно из лучших качеств языка VBA заключается в том, что он имеет достаточно много общего с обычным английским языком. Правда, выглядит он как довольно ломаный английский, но не волнуйтесь— как раз в этой книге я постараюсь обучить вас свободному владению языком, с помощью которого пишется VBA-код.

Что визуального в Visual Basic для приложений?

К счастью, VBA во многом избавляет от необходимости нудного печатания программного кода. В одних случаях вы записываете команды, которые нужны в приложении, и используете их в качестве отправной точки при создании новой программы.

В других случаях VBA позволяет создавать части программы, которые отвечают за появляющиеся на экране объекты (диалоговые окна, кнопки, флажки, переключатели и т.п.), просто рисуя их с помощью мыши или выбирая опции из диалоговых окон. Именно это позволяет назвать VBA визуальным программным средством. С помощью VBA вы можете построить экранное представление своей программы, почти не затратив на это времени. Если бы вам

пришлось определять те же элементы и программировать их появление на экране вручную, вам потребовались бы на это часы, дни, а может, и годы.

К сожалению, VBA не может заранее угадать, что должна *делать* ваша программа. Вам все равно придется печатать программный код, определяющий функции кнопок, флажков и переключателей в вашей программе. И даже если в программе нет ни кнопок, ни переключателей, вам понадобится напечатать программный код для тех шагов, которым программа должна следовать при ее выполнении.

Главные события

В VBA событие означает нечто, что случается при выполнении программы, если необходимо изменить ее ход. Самый простой пример— щелчок кнопкой мыши. Когда пользователь вашей программы щелкает на кнопке с надписью Вычислить возраст, это действие является событием. Точно так же всегда случается событие, когда пользователь нажимает клавишу на клавиатуре, дважды шелкает кнопкой мыши или просто двигает мышь.

Конечно же, ваша программа не может отвечать на все эти события с помощью телепатии. Напротив, для каждого события, которое должна распознавать программа, вам нужно самому написать *процедуру обработки события,* представляющую собой специальную порцию программного кода VBA. Вам потребуется одна процедура обработки события, чтобы отвечать на щелчки на кнопке Вычислить возраст, но потребуется еще одна, если вы сочтете необходимым выполнять какие-либо действия и тогда, когда указатель мыши просто помещается на кнопку без щелчка. Ясно, что, если у вас есть еще кнопка Солгать про возраст, потребуется дополнительная процедура обработки события для щелчка на этой кнопке. Тема создания процедур обработки событий подробно обсуждается в главе 10.

Цикл создания программы

Независимо от используемых программных средств, процесс создания новой программы можно разбить на пять простых шагов.

1. Проектирование.
   Здесь определяется, что должна делать программа, как она должна выглядеть на экране и взаимодействовать с другим программным обеспечением.
2. Реализация.
   Для успешного выполнения п. 1 конструируются окна и другие элементы программы, появляющиеся на экране, и создается необходимый программный код. Это совсем несложно.
3. Тестирование.
   Запускают программу, чтобы проверить, выглядит ли она так, как нужно, и выполняет ли то, что нужно. Как правило, не выглядит и не выполняет.
4. Отладка.
   Весь программный код тщательно проверяется до тех пор, пока не выяснится причина неправильного поведения и не будут внесены соответствующие коррективы (как всякое приличное средство программирования, VBA имеет специальные возможности, облегчающие процесс отладки).
5. Повторное тестирование.
   Повторяют пп. 3 и 4 до тех пор, пока хватит сил и терпения.

К счастью, специально заучивать эту последовательность шагов не придется — они станут, если можно так выразиться, вашей второй натурой, как только вы всерьез займетесь программированием в VBA.

Подробнее о том, что можно сделать в VBA

Выше, в разделе "Зачем мучиться с VBA?", я уже упоминал о том, как с помощью VBA можно усовершенствовать имеющиеся приложения. Пришло время обсудить это подробнее, чтобы вам стало ясно наконец, что же все-таки можно получить от VBA.

Прежде чем приступить к реализации наших совместных планов грандиозных усовершенствований, замечу следующее: то, что можно получить от VBA, зависит от приложения, с которым вы работаете. Некоторые программы позволяют переопределить практически каждый элемент пользовательского интерфейса и каждую встроенную команду, предоставляя тем самым в изобилии строительный материал, из которого вы можете конструировать новые возможности, В других приложениях выбор оказывается намного скуднее.

Настройка пользовательского интерфейса

Одна из наиболее очевидных ситуаций, где стоит использовать VBA, — это изменение пользовательского интерфейса приложения, чтобы он стал привычнее и удобнее для вас. (На всякий случай замечу, что *пользовательский интерфейс* означает то, как программа выглядит на экране, как в ней работают мышь, клавиатура и другие всевозможные средства, предоставленные программой вам, пользователю, для взаимодействия с программным обеспечением.) Во многих приложениях VBA позволяет изменять элементы пользовательского интерфейса так, как это удобно. С помощью всего нескольких строчек программного кода вы можете добавлять, удалять или изменять взаимное расположение кнопок на панелях инструментов, создавать новые панели инструментов, переопределять раскладку клавиатуры или модифицировать структуру меню. На рис. 1.1 показано окно Microsoft Word с любезно измененным VBA пользовательским интерфейсом.

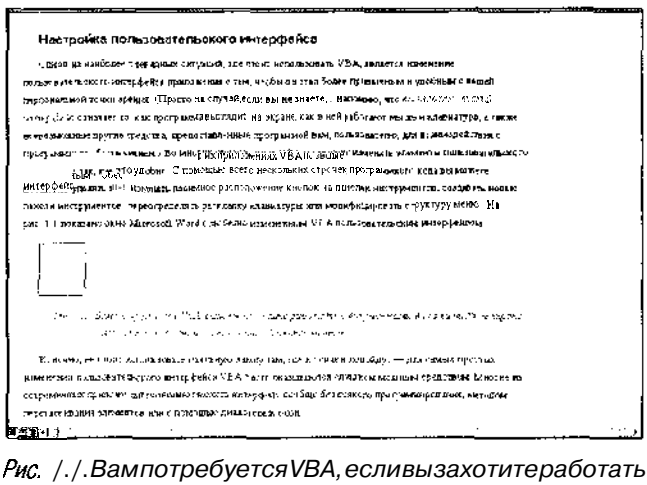


Рис. 1.1. Вам потребуется VBA, если вы захотите работать с документами Word на чистом экране, совершенно свободном от всех панелей инструментов

Конечно, не стоит использовать паяльную лампу там, где и спички подойдут,— для самых простых изменений пользовательского интерфейса VBA часто оказывается слишком мощным средством. Многие из современных приложений позволяют изменять интерфейс вообще безо всякого программирования — с помощью перетаскивания элементов или диалоговых окон.

Тем не менее VBA полезен и в таких приложениях. С одной стороны, даже исключительно дружественные в этом смысле приложения могут не изменять некоторые части своего интерфейса без помощи VBA. С другой стороны, что еще важнее, VBA позволяет организовать целые группы автоматических изменений пользовательского интерфейса прямо во время работы — в зависимости от того, что вы делаете с программой в данный момент.

Предположим, вы работаете с суммами, превышающим миллион долларов, постоянно используя при этом одни и те же три панели инструментов. С помощью VBA вы можете создать программу, которая станет открывать эти панели инструментов вместе, а затем одновременно прятать их, когда в них нет необходимости.

До сих пор я говорил об изменении пользовательского интерфейса VBA-приложения, но не о создании пользовательского интерфейса самой VBA-программы. Хотя многие VBA-программы и представляют собой автоматы, выполняющие работу безо всякого вмешательства с вашей стороны, неизбежно наступит момент, когда вам понадобится создавать свои диалоговые и другие окна (для них в VBA используется обобщающий термин *формы).*

Добавление новых возможностей

VBA оказывается как раз кстати, когда необходимо расширить функциональные возможности приложения. Предположим, вы пожелали, чтобы в первый понедельник каждого месяца на всех создаваемых вами компьютерных иллюстрациях появлялись красные звездочки, а в каждый второй вторник— синие сердечки. Вправе ли вы ожидать, что разработчики графического приложения предусмотрели ваши пожелания? Используя строительные блоки, предоставленные в ваше распоряжение разработчиками данного приложения, вы создадите VBA-программы, которые приблизят вас к мечте.

В самом простом случае VBA-программа просто содержит определенную последовательность команд, выполняя одну и ту же последовательность шагов при каждом запуске. Тем не менее даже в этом случае VBA-программа экономит ваше время по сравнению с необходимостью активизировать каждую команду последовательности вручную.

Например, в Microsoft Word нет команды для сохранения выделенного блока текста в отдельном файле. Для решения этой задачи вам придется скопировать выделенный текст в буфер обмена, создать новый документ, скопировать в него содержимое буфера, сохранить и закрыть этот новый документ. С помощью VBA можно создать небольшую программу, которая выполнит все эти шаги автоматически (тем самым обеспечив пользователю Word возможность, которую большинство других текстовых процессоров имеют еще со времен CP/M, — операционной системы, использовавшейся на самых первых персональных компьютерах). VBA-код такой программы может выглядеть примерно так:

```
Public Sub CopyBlockToFile()
Selection.Range.Copy                    ' копирует выделенное
                                        ' в буфер обмена
Documents.Add                           ' создает новый документ
Selection.Range.Paste                   ' вставляет содержимое
                                        ' буфера
                                        ' в новый документ
```

```
Dialogs(wdDialogFileSaveAs).Show        ' открывает диалоговое
                                        ' окно Сохранение документа
ActiveDocument.Close                    ' закрывает новый документ
End Sub
```

Но, поскольку VBA полноценный язык программирования, VBA-программа совсем не обязана следовать в точности одним и тем же шагам при каждом запуске. Напротив, она должна всякий раз действовать в соответствии с теми условиями, которые обнаруживает после своего запуска. Вот несколько примеров того, как это должно выглядеть.

- ✓ Во время запуска VBA-программа может отвечать на то, что делает в данный момент приложение, или на конкретные установки системы в целом. В случае упомянутого примера с Word, можно потребовать, например, чтобы программа сохраняла новый документ, если только некоторый текст *уже* выбран, а иначе ей предписано только чесать затылок и поглаживать животик. Подобным образом вы можете пожелать, чтобы ваша программа выполняла одни действия утром, а другие — вечером.
- ✓ VBA-программа может (в зависимости от содержимого текущего документа) выполнять определенные действия с другим документом того же или другого приложения.
- ✓ VBA-программа может отвечать на информацию, вводимую пользователем во время работы программы (рис. 1.2). С помощью VBA легко создавать диалоговые окна, предлагающие пользователю либо выбрать из списка заранее определенных возможностей, либо ввести некоторый текст или числовые значения.
- ✓ VBA-программа может менять свое поведение в зависимости от текущего знака Зодиака или фазы Луны. (Здесь, к сожалению, вам еще придется написать программный код, осуществляющий перевод даты и времени в соответствующие знак Зодиака и фазу Луны.)

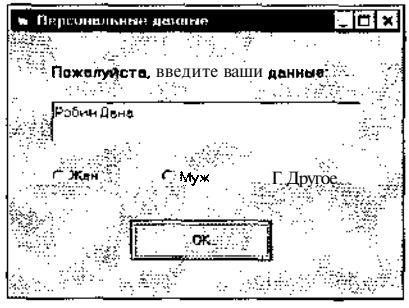


Рис. 1.2. Если для продолжения работы программы требуется определенная информация от пользователя, можно создавать диалоговые окна, подобные этому

Как *практически* заставить VBA-программы реагировать на изменение условий, мы подробно обсудим в следующих главах.

Создание более сложных программ

Создавать VBA-программы, добавляющие функциональные возможности существующим приложениям, может каждый, кто способен запомнить свое школьное расписание или прочитать за завтраком инструкцию на упаковке с вермишелью быстрого приготовления.

Но вам будет не лишним знать, что совсем не обязательно ограничиваться минимальными изменениями и усовершенствованиями существующих приложений; VBA позволяет создавать полноценные и в высшей степени сложные программы, способные конкурировать с мощью готовых продуктов, имеющихся на рынке приложений.

Пользовательские программы, созданные с помощью VBA, интенсивно используют функциональные возможности того VBA-приложения, на которое они опираются (пользовательская VBA-программа сумеет использовать функциональные возможности из *нескольких* VBA-приложений одновременно). Однако перед пользователем такая программа предстает как цельный самостоятельный объект со своим собственным набором окон, кнопок и меню.

Создание полномасштабной пользовательской программы — штука достаточно амбициозная. Но эта книга предлагает практически всю необходимую для этого информацию, в том числе и взодное описание способов одновременного доступа из программы к нескольким VBA-приложениям.

Кстати, VBA не обязывает вас использовать функциональные возможности приложения. При желании совершенно игнорируйте их, используя только созданные самостоятельно. Но для запуска даже такой пользовательской программы все равно потребуется VBA-приложение (между прочим, в Visual Basic такой проблемы нет). Здесь вас подстерегает еще одна неприятность, но о ней мы поговорим позже, в подразделе "Почему VBA обычно медленнее, чем Visual Basic".

Совсем не тот BASIC

Можно сказать, что VBA является прямым наследником оригинального языка программирования, получившего имя BASIC. Так сказать, безусловно, можно, но смысла в этом примерно столько же, сколько в утверждении, что эволюция человека восходит к амебе — уж слишком многое с того времени изменилось.

Оригинальная версия языка BASIC была создана в 1960-х годах в надежде облегичить программирование для широкой аудитории. BASIC расшифровывается как Beginner's All-purpose Symbolic Instruction Code. По сравнению с такими языками программирования, как C++ или FORTRAN, команды языка BASIC более сходны с фразами обычного английского языка, поэтому эти команды проще понять и запомнить.

Многие из "слов" специального назначения, используемые в VBA, присутствовали уже в оригинальном языке BASIC и имели там сходные функции. Точно так же "грамматика" VBA — определенные правила и порядок, в соответствии с которыми слова образуют выражения, — берет свое начало в языке BASIC.

Однако язык VBA значительно эволюционировал по сравнению с ранними версиями языка BASIC, оставив их далеко позади. Многие из команд VBA, а также правила их применения в языке BASIC вообще отсутствуют. Проще говоря, язык VBA может больше — и значительно больше, — чем старый добрый BASIC, особенно в том, что касается вывода на экран всевозможных интересных форм и взаимодействия с другими приложениями.

Кроме того, VBA — это не только язык программирования, он включает также полноценную *интегрированную среду разработки* с полным набором специализированных окон, призванных помочь вам в проектировании, отладке и тестировании программ (читайте ниже подраздел "Интегрированная среда разработки приложений VBA"). VBA имеет еще кое-что из того, о чем BASIC не мог и мечтать,— позволяет создавать замысловатый пользовательский интерфейс без необходимости печатания программного кода вручную. Как уже упоминалось, именно это и превращает VBA в *визуальное* средство разработки приложений.

Интегрированная среда разработки приложений VBA

Вся работа с VBA происходит в интегрированной среде разработки. Этот термин может выглядеть холодно и пугающе, но не давайте ему себя запугать, — вы должны воспринимать интегрированную среду разработки как уютный дом, где можно выполнить всю программистскую работу в тепле и комфорте.

Интегрированная среда разработки, предлагаемая VBA, представлена как *редактор Visual Basic*. Редактор Visual Basic представляет собой окно приложения с одним меню и набором панелей инструментов, в котором вы получаете доступ к целому ряду дочерних окон, обеспечивающих возможность использования всех средств, необходимых для создания программ. На рис. 1.3 представлена версия редактора Visual Basic, предлагаемая Visio, но выглядит она точно так же, как и те, которые предлагаются в других VBA-приложениях.

Не все VBA-приложения имеют встроенный редактор Visual Basic (подробнее об этом речь идет ниже, в подразделе "Другие "диалекты" VBA").

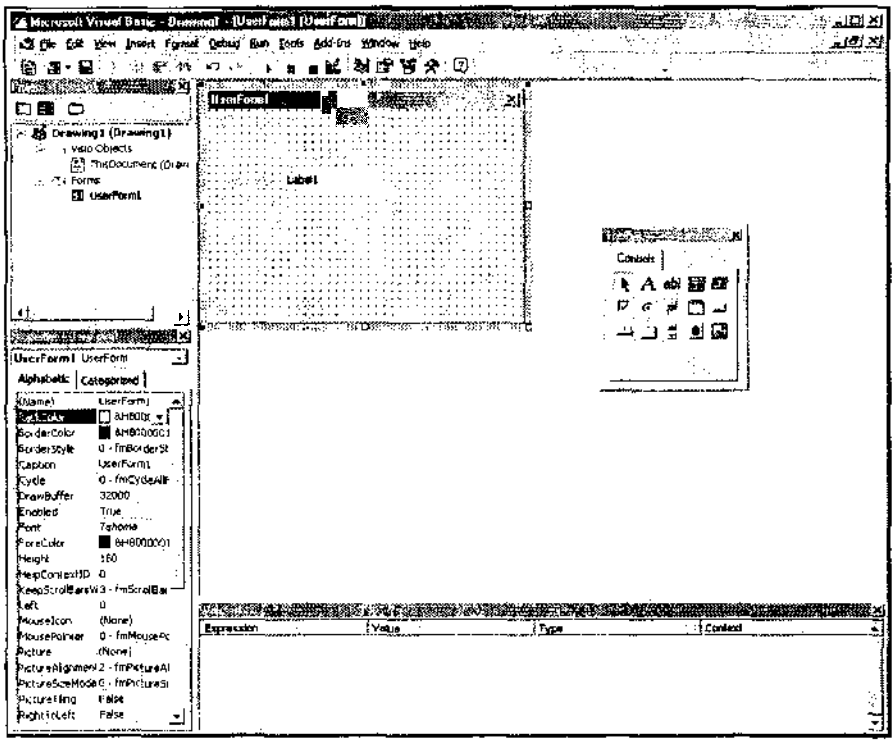


Рис. 1.3. Внешний вид редактора Visual Basic в Visio

Подробное описание редактора Visual Basic вы найдете в главе 5 этой книги. Пока же ознакомьтесь со следующим списком некоторых элементов экипировки этого редактора, что ласт вам хотя бы поверхностное представление о том, что он предлагает.

- ✓ Место для проектирования *форм* (диалоговых и других окон), которые будут использоваться в создаваемой вами программе.

- ✓ Панель, из которой можно выбирать кнопки, флажки и другие элементы управления, которые вы пожелаете поместить в формы (читайте ниже подраздел "Элементы управления ActiveX и другие").
- ✓ Окно, с помощью которого можно сообщить VBA, как должны выглядеть и что должны уметь делать формы и элементы управления на них.
- ✓ Окна для ввода и редактирования программного кода VBA, с помощью которого вы можете заставить свою программу делать что-нибудь полезное.
- ✓ Окна с текущими значениями *переменных* — числа и текст, которые в них хранятся и меняются, — во время выполнения программы (эти окна значительно облегчают жизнь, когда наступает сезон охоты на спрятавшиеся в программе ошибки).

Макросы и VBA

Чаще других используется способность VBA объединять в группу последовательность различных команд приложения. Если приходится часто использовать одни и те же команды в одной и той же последовательности, имеет смысл сохранить эту последовательность в виде VBA-программы. В результате вы сможете активизировать всю эту последовательность одной командой — той самой, которая запускает созданную вами с этой целью VBA-программу.

Между макросом, который создается в процессе записи, и VBA-программой, код которой вы печатаете вручную, нет никакой — вообще никакой • — принципиальной разницы, если, конечно, не считать сам способ создания. Вы можете редактировать программный код макроса, добавляя в него или удаляя из него элементы точно так же, как будто вы напечатали код макроса своими собственными руками.

Создать VBA-программу этого типа проще всего с помощью непосредственной записи всей последовательности выполняемых команд. Для этого большинство VBA-приложений предлагает *средство записи макросов,* которое работает подобно магнитофону. С того момента, как вы даете указание начать запись макроса, начинают записываться все команды, которые вы используете в приложении. После того как вы остановите запись, средство записи макросов конвертирует записанные команды в эквивалентные им строки VBA-кода. Полученная в результате этого VBA-программа имеет специальное название — *макрос.* Все тонкости процесса записи макросов обсуждаются в главе 2.

Элементы управления ActiveX и другие

В любой программе для Windows элементы управления — это все те бедолаги на экране, на которых можно щелкнуть или что-нибудь напечатать, чтобы вызвать определенное ответное действие программы. Среди самых распространенных элементов управления следует упомянуть кнопки па панелях инструментов и в диалоговых окнах, переключатели и флажки, предназначенные для выбора заранее определенных опций, и текстовые поля, в которых можно ввести или изменять данные. Пример диалогового окна со множеством различных элементов управления показан па рис. 1.4.

VBA содержит все эти. а также вообще все стандартные для Windows типы элементов управления. Работают они подобно подключаемым компонентам. Чтобы добавить элемент управлений в любую из форм (например, в диалоговое окно), щелкните на соответствующем

элементе управления в панели элементов управления (специальной панели инструментов, пример которой показан на рис. 1.5), а затем щелчком на форме "прилепите" этот элемент к форме.

Благодаря технологии, загадочно называемой *ActiveX,* можно не ограничиваться элементами управления, предлагаемыми VBA. Разработанная тоже *Microsoft*, ActiveX определяет стандарт, в соответствии с которым разработчики программного обеспечения создают взаимозаменяемые элементы управления, которые вы можете при желании подключать к своим программам. Элементы управления, предлагаемые VBA, являются элементами управления ActiveX, но, кроме них, есть еще очень много других. Элементы управления ActiveX работают не только в VBA-программах, но и в программах, созданных с помощью C++ или Java. Чтобы получить возможность использовать в своих VBA-формах новый элемент управления ActiveX, достаточно добавить его в панель элементов управления (за инструкциями обратитесь к главе 14). После этого с ним можно обращаться точно так же, как и с элементами управления, изначально присутствующими в VBA.

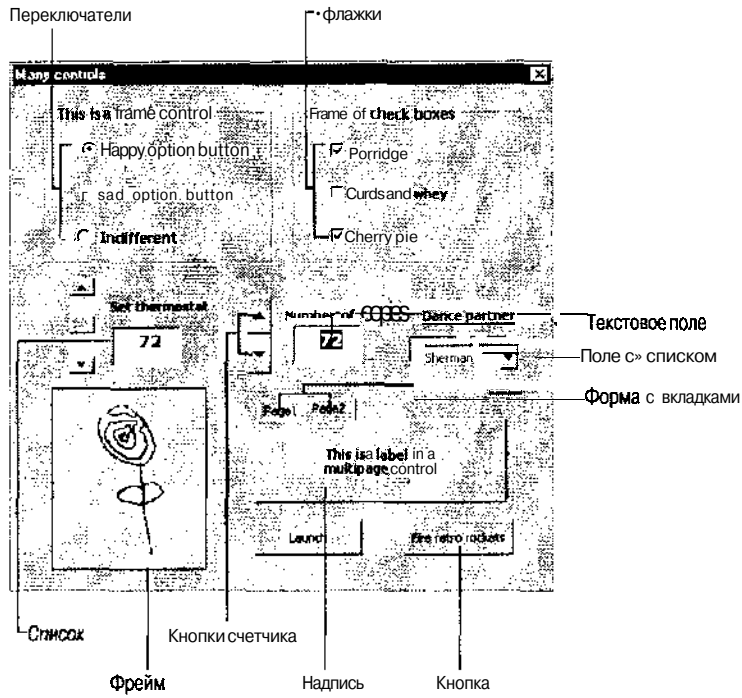


Рис. 1.4. Разнообразные элементы управления в диалоговом окне

Рис. 1.5. Панель элементов управления VBA предлагает целый ряд элементов управления для ваших форм

Любой, кто пожелает, может создавать такие специализированные программные заготовки для своих собственных нужд или же для продажи таким программистам, как вы. Огромные размеры рынка программ для Windows объясняют, почему так велико количество пред-

лагаемых элементов управления ActiveX. Среди них можно обнаружить как усовершенствованные варианты стандартных элементов управления, так и элементы управления совершенно новых типов, например круговые и линейные шкалы, часы, календари и многое другое. Если VBA не предлагает нужный вам элемент управления, почти наверняка в этом разнообразии вы сможете найти подходящий и купить его. Представление о том, как выглядят элементы управления, можно получить из рис. 1.6.

Для некоторых из элементов управления ActiveX, предлагаемых различными производителями, главным достоинством является их внешний вид. Эти элементы управления функционально более или менее идентичны стандартным, но выглядят лучше. Представьте себе, что вы собираетесь создать приключенческую игру на космическую тему и хотели бы обеспечить игрокам возможность регулировки скорости своих межгалактических лайнеров. Знаю я, знаю — нет таких людей, кто собирается использовать VBA для создания приключенческих игр! Но представьте только на минуту — разве не лучше вместо пары унылых стрелочек, предлагаемых счетчиком, предложить игрокам большие и яркие изображения космических ракет?

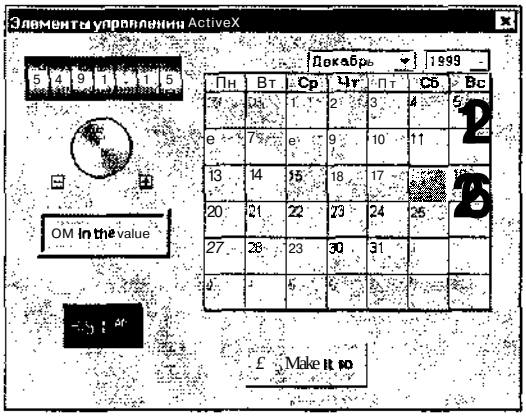


Рис. 1.6. В этой форме размещены элементы управления, которые не предлагаются стандартно VBA

Но во многих случаях предлагаемые элементы управления ActiveX могут делать то, чего не могут стандартные, например размещать на формах небольшие электронные таблицы с данными. Краткий обзор самых популярных и полезных элементов управления ActiveX различных поставщиков вы найдете в главе 14.

Вы можете даже самостоятельно создавать свои собственные элементы управления ActiveX. Для этого используют Visual Basic 5 или 6 (либо профессиональную версию, либо версию для предприятий),

Объектно-ориентированное программирование и VBA

В результате своей эволюции Visual Basic превратился в (почти) *объектно-ориентированный* язык программирования. Хотя освоение приемов работы с объектами и представляет некоторую трудность в начале знакомства с VBA, ожидаемая отдача того стоит. Тонкости программирования объектов будут рассмотрены в главе 12, но уже здесь имеет смысл получить хотя бы поверхностное представление о предмете.

VBA как стандарт (точнее, род стандарта)

Одно из главных преимуществ VBA — то, что это *стандарт,* т.е. он работает совершенно одинаково во всех VBA-приложениях. Если вы научитесь создавать VBA-программы, скажем, в Microsoft Word, все ваши знания и умения точно так же будут применимы в Visio, AutoCAD и любом другом VBA-приложении.

В таком утверждении доля правды действительно велика, Редактор Visual Basic и все его составляющие идентичны во всех VBA-приложениях (точнее, во всех, где есть редактор Visual Basic). Работает редактор Visual Basic тоже везде одинаково. Это касается, например, процесса создания форм (опять напомню, что формы — это диалоговые и другие окна, создаваемые вами для своих программ). Вы можете создать форму в одном VBA-приложении и использовать ее затем в другом. Хотя, честно говоря, проектировать и создать формы в VBA совсем просто. Но когда дело доходит до языка программирования, составляющего ядро VBA, там уже все далеко не так согласованно. С одной стороны, базис языка всегда один и тот же. Например, для определения переменной всегда можно использовать специальное слово `Din` (напомню, что переменная — это место, где программа хранит число, кусочек текста или нечто другое, что изменяется в процессе выполнения программы). Так, в любом VBA-приложении выражение `Dim MyVariable as Integer` создает переменную с именем `MyVariable` и определяет ее как целую, а не как текст или число с десятичными знаками.

Точно так же одинаково определяются выполняющие конкретные задачи подразделы программы *{процедуры* и *функции),* независимо от того, в каком из VBA-приложений вы это делаете.

Однако, помимо этих основополагающих команд, в каждом из VBA-приложений довольно много VBA-команд, уникальных именно для этого приложения. Это значит, что вам придется достаточно многое выучить дополнительно, если вы захотите перейти от программирования в рамках одного VBA-приложения к программированию в рамках другого, — хотя и предполагается, что VBA стандартизирован.

Конечно, совсем не удивительно, что такое специализированное на графических изображениях приложение, как Visio. имеет команды, отличные от команд текстового процессора Word. Печально то, что различные VBA-приложения часто используют разные команды даже для тех возможностей, которые у этих приложений общие. Ярким примером такой несогласованности служат команды настройки меню. Можете ли вы поверить, но даже Word, Excel и PowerPoint — эти вечно целующиеся кузины из семейства Microsoft Office — для настройки некоторых элементов имеют совершенно разные VBA-команды. Ох, нет совершенства в этом мире!

Строго говоря, сам язык VBA остается одним и тем же во всех VBA-приложениях. Он кажется меняющимся, потому что в каждом из приложений есть свой уникальный набор *объектов* с их собственными методами (командами) и свойствами (характеристиками). Подробнее об объектах, методах и свойствах говорится в главе 12.

VBA 5 против VBA 6

Эта книга охватывает последние толкование VBA — версию 6, которая используется в приложениях Office 2000, а также в продуктах iGrafx фирмы *Micrografx,* Corel WordPerfect

2000 и CorelDraw, VBA 5 вышел из Microsoft Office 97, и большинство VBA-приложений третьих фирм поддерживают именно эту версию VBA. Различия между этими двумя версиями не слишком существенны, так что вы можете практически безболезненно перейти от любой из этих двух версий к другой. И все же вам не помешает ознакомиться со следующим списком наиболее важных отличий.

- ✓ Справочная система VBA 5 основана на использовании старой программы WinHelp, в то время как VBA 6 использует для той же цели новую технологию HTML Help фирмы *Microsoft* (между прочим, многие до сих пор предпочитают WinHelp).
- ✓ В VBA6 ядро языка VBA пополнили 13 новых функций, предназначенных главным образом для манипуляций с текстом и форматирования значений. Описание встроенных функций VBA с указанием тех из них, которые доступны только в VBA 6, вы найдете в главе 11.
- ✓ Формы в VBA 6 могут быть немодальными, в отличие от VBA 5, где все формы — модальные. Здесь я даже не пытаюсь определить эти термины — они подробно обсуждаются в главе 10.

Еще несколько усовершенствований в VBA 6 пригодятся только опытным разработчикам и поэтому здесь не обсуждаются, поскольку выходят за рамки этой книги.

Другие "диалекты" VBA

Большинство VBA-приложений имеют "полный фарш" — и средство записи макросов, и полноценный редактор Visual Basic, и полную версию языка программирования VBA. К таким "Кадиллакам" VBA-приложений можно отнести Visio, WordPerfect, AutoCAD и, безусловно, "тяжелую артиллерию" от *Microsoft* — Word, Excel и PowerPoint.

Microsoft Access немного выпадает из общего ряда VBA-приложений. В Access нет средства записи макросов, поэтому программный код макросов придется записывать вручную, — по этой причине они не считаются VBA-макросами. Формы Access (пользовательские диалоговые окна) не переносятся в другие VBA-приложения. И в то время как Access 2002 содержит редактор Visual Basic, в Access 97 его нет. Вместо редактора там вы получаете отдельные окна для печатания VBA-кода, отладки и проектирования форм.

В некоторых приложениях (например, в Internet Explorer и Outlook) используется урезанная версия VBA, называемая *VBScript*. Такие VBScript-приложения не имеют редактора Visual Basic, ла и сам язык программирования не полон. Но все равно этот язык остается достаточно мощным.

VBA против Visual Basic

Помимо того, что VBA не позволяет вам создавать отдельные приложения, одно из основных отличий состоит в том, что программы, написанные на VBA, работают медленнее, чем программы, написанные на Visual Basic.

Почему VBA обычно медленнее, чем Visual Basic

Кроме того, что вы не можете создавать в VBA отдельные, автономно работающие приложения, между VBA и Visual Basic есть еще одно серьезное отличие — VBA-программы обычно выполняются медленнее.

Причина этого обнаруживается просто: VBA-программу приходится *компилировать* хотя бы раз в каждой сессии работы с соответствующим VBA-приложением. Компиляция — это процесс перевода *программного кода* (который вы можете прочитать) в *машинный код* (представляющий собой инструкции, которые непосредственно исполняются компьютером во время выполнения программы). *Компилятор* представляет собой программу, которая переводит программный код, понятный человеку, в инструкции, которые может выполнить компьютер.

После выполнения компиляции программы, созданной в Visual Basic, она сохраняется на жестком диске компьютера в форме машинного кода, который всегда может быть выполнен компьютером с максимальной скоростью, как только программа будет загружена в память, В противоположность этому, VBA-программы хранятся в виде VBA-кода. Когда вы даете указание выполнить VBA-программу, происходит следующее.

I. **Выполняется компиляция программы.**
   Именно этот шаг является узким местом для всего процесса. Пока компилятор VBA выполняет свою работу, слышно, как журчит жесткий диск, но создается впечатление, что больше ничего не происходит.

1. **Полученная в результате компиляции версия программы сохраняется в памяти компьютера.**
2. **Наконец-то! Программа начинает выполняться.**

Шаг 2 дает надежду на сведение потерь к минимуму. До тех пор пока скомпилированная версия VBA-программы хранится в памяти, эта программа может выполняться с максимальной скоростью всякий раз, когда понадобится. Но как только вы выйдете из VBA-приложения, эта скомпилированная программа без следа растворится в эфире. Еще хуже то, что скомпилированная версия VBA-программы может лишиться своего временного жилья в памяти задолго до того момента, когда вы решите покинуть приложение. Такое ее изгнание случится, если программа в какой-то момент не использовалась, и как раз тогда системе потребовалась дополнительная память, например для запуска других приложений.

Что касается тех задач, на которых специализируется VBA (а к ним относится работа с объектами "базового" VBA-приложения), то они будут выполняться программой, созданной в VBA, действительно быстрее, чем такой же программой, написанной в Visual Basic.

Различия между VBA и Visual Basic

VBA имеет очень много общего с Visual Basic, своим старшим братом, предназначенным для создания независимых приложений. А раз языки похожи, вы можете перенести большую часть своих навыков в программировании на VBA в Visual Basic. Однако вам следует помнить о некоторых очень важных различиях.

К концу 2001 года ожидается перевоплощение Visual Basic в Visual Basic.NET. Хотя изменения в основном направлены на расширение возможностей и упрощение использования, сам язык будет отличаться от VBA. Поэтому вам придется изменить некоторые приемы программирования, принятые в VBA, чтобы они сработали и в Visual Basic.NET. Конечно же, вы можете ограничиться Visual Basic версий 5 и 6 — инструментами, которые будут работать еще не один год и которые используют один базовый язык программирования с VBA. Однако вы должны иметь в виду, что VBA и Visual Basic используют разную систему создания и отображения форм.

Глава 2

Не пишите программу, когда можно записать макрос

В этой главе...

- Запись макросов — последовательностей команд, чтобы пользоваться ими снова и снова
- Запуск средства записи макросов
- Тонкости записи макросов: они должны работать так, как вы планировали
- Просмотр и редактирование программного кода макроса в редакторе Visual Basic

Зачем мучиться с набором программного кода, если в этом нет необходимости? Если все, что вам нужно, — это автоматизировать выполнение некоторой последовательности команд вашего приложения, запишите их в виде *макроса.* Прочтите эту главу, чтобы узнать, как с помощью записи макросов минимизировать объем программного кода, который придется набирать вручную.

Как работают макросы

Некоторые VBA-приложения — среди них Microsoft Word, Excel и PowerPoint— имеют *средство записи макросов,* работа которого напоминает работу обычного магнитофона. После включения средства записи макросов оно будет записывать все команды, которые вы используете в приложении, до тех пор, пока не щелкнете на кнопке Остановить запись. После того как макрос записан, вы сможете воспроизводить его каждый раз, когда вам потребуется записанная в нем последовательность команд.

Кстати, слово *макрос* означает большой. Подразумевается, что вы объединяете множество небольших команд в одну большую, На самом же деле, ваш макрос может быть таким маленьким и скромным, каким вы пожелаете, — если у вас слишком много времени, можете создать себе целый ряд бесполезных макросов, включающих всего по одной команде.

В VBA-приложениях, *не* имеющих средства записи макросов, термин *макрос* может означать любую созданную вами VBA-программу. Например, в Visio версий 4.5 и 5 указание создать макрос открывает окно создания новой программы в редакторе Visual Basic. С другой стороны, как минимум в одном из основных VBA-приложений — в Access — макрос вообще напрямую никак не связывается с VBA. В Access, хотя и можно создавать макросы, содержащие последовательности команд, они не сохраняются в виде VBA-кода автоматически (подробности вы найдете ниже во врезке "Макросы — не росы"). Средство записи макросов отсутствует во всех версиях Access — там есть специальное окно для выбора команд, которые вы собираетесь поместить в макрос.

Макросы — не росы

Макрос - это просто еще одно название для VBA-программы. Средство записи макросов во время своей работы конвертирует каждую из используемых вами команд в соответствующие строки программного кода VBA. Законченный макрос сохраняется как *процедура* VBA. Процедуры, как разъясняется Б главе 6, - это отдельные единицы программного кода VBA, которые можно вызвать по имени для выполнения. (Если уж говорить совсем строго, то макрос представляет собой процедуру типа Sub, не имеющую аргументов. Вам нужна именно такая строгость?)

Еще один теоретический момент: макросы есть и в Access, и в VBA, но в Access они не являются VBA-программами. Макросы в Access создаются с помощью последовательности команд в специально предлагаемом для этого диалоговом окне. Боюсь еще больше запутать дело, но я должен тут добавить, что макрос в Access может вызывать VBA-процедуру Access. Вот так-то.

Когда следует записывать макрос

Макросы экономят время и снимают раздражение. Это действительно так. Компьютер воспроизведет последовательность команд куда быстрее, чем это сделаете вы, щелкая на соответствующих кнопках команд и выбирая пункты соответствующих меню. При этом компьютер не допустит ни единой ошибки. И ваше настроение, несомненно, улучшится, поскольку люди обычно не любят повторять одно и то же больше двух-трех раз подряд.

Поэтому проверьте, не приходится ли вам снова и снова повторять в приложении одни и те же последовательности команд. Как только вы обнаружите, что такие последовательности у вас есть, сразу же запишите их в виде макроса. А еще лучше, если вы знаете наперед, что какую-то новую последовательность команд вам придется использоваться и в дальнейшем, запишите соответствующий ей макрос уже при первом случае ее применения. И уже со следующего раза используйте макрос.

Макрос — и вещь хорошая, и развитие стимулирует

Макросы очень полезны для своего прямого назначения, но не менее полезны они для использования программного кода записанного макроса как примера для начала освоения программирования в VBA. После того как вы запишете несколько макросов, вы наверняка загоритесь желанием как-нибудь их улучшить и сделать более гибкими. 3 результате совсем небольших усилий с помощью VBA вы можете добавить своим макросам немного "интеллекта" и заставить их выполнять различные действия в зависимости от ситуации. Это будет обсуждаться ниже, в разделе "Редактирование макросов как способ создания программного кода".

Планирование макросов

Прежде чем вы пропустите этот тоскливый раздел и с головой погрузитесь в процесс записи макросов, выслушайте один довольно консервативный совет: чтобы избежать лишней головной боли, уделите немного времени планированию макроса перед тем, как записывать его.

Можно, конечно, запустить средство записи макросов и начать громоздить одну за другой команды сразу, как только в голове появится хотя бы смутное представление о

том, что должен делать ваш новый макрос. Вот только средство записи макросов очень добросовестно запишет буквально каждую из вызываемых вами команд, в том числе и все допущенные вами ошибки. А в восьми или девяти случаях из десяти такие ошибки у вас будут.

Ясно, что можно удалить неудавшийся макрос и начать все сначала. Но можно и избавить себя от таких проблем, если потратить немного времени на планирование макроса. Продумайте те шаги, которые должен выполнять макрос, предварительно попробуйте их, чтобы убедиться, что результат будет соответствовать запланированному, и только затем записывайте макрос.

При планировании макроса учитывайте все специальные условия, необходимые для его правильной работы. Процесс записи должен включать и добавление в макрос команд, нужных для воссоздания этих необходимых условий.

Представьте, например, что вы работаете в Microsoft Word. Скажем, требуется, чтобы макрос вставлял текущую дату в начало каждого документа. Можно, конечно, предположить, что текстовый курсор при этом уже находится в самом начале документа. Но не следует быть столь самоуверенным. Даже если курсор находится в нужном месте, когда вы начинаете записывать макрос, все равно первой записываемой командой должна быть команда перемещения курсора в начало документа. Только тогда макрос будет работать правильно, независимо от того, где окажется текстовый курсор.

Запись макросов

К счастью, записывать макросы так же просто, как ложиться в постель. Тем более, что я собираюсь разложить здесь вам все по полочкам.

Запуск средства записи макросов

Запустить средство записи макросов можно одним из следующих способов.

- ✓ Выбрать из меню Сервис⇨Макрос⇨Начать запись. В последних версиях Office меню **Макрос** может не появляться до тех пор, пока вы не щелкнете на кнопке с направленной вниз стрелкой внизу в меню Сервис.
- ✓ В панели инструментов щелкнуть на кнопке **Начать запись макроса,** если у вашего приложения таковая есть. Например, в Microsoft Office кнопка **Начать запись макроса** расположена в панели инструментов Visual Basic.

В результате любого из этих действий появится диалоговое окно **Запись макроса.** На рис. 2.1 показано диалоговое окно **Запись макроса** из Microsoft Excel (это окно в разных приложениях выглядит по-разному).

Как выбрать имя для малютки

Стараясь быть полезным, диалоговое окно **Запись** макроса предложит самое оригинальное из подходящих имен для вашего нового макроса — Макрос1 (или Макрос2, или Макрос3). Вежливо поблагодарив, быстренько удалите предлагаемое имя и впечатайте вместо него нечто более вам подходящее.

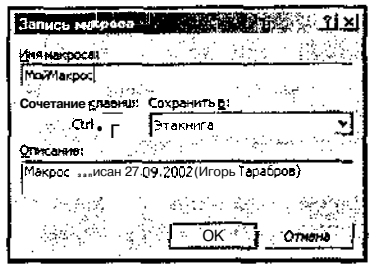


Рис. 2.1. Диалоговое окно Запись макроса в Excel

Имена макросов должны, конечно, ассоциироваться **с** выполняемыми ими задачами, Однако при этом следует придерживаться определенных правил. Ну-ка, повторяйте все хором за мной.

- ✓ **Имя макроса должно начинаться с буквы, а не** с **цифры.** А вот после первой буквы уже могут идти и цифры.
- ✓ **Имя макроса не должно содержать пробелов.** Для выделения начала слов в имени макроса следует использовать прописные буквы, например, как здесь:
- ✓ `ОчисткаВсегоЖесткогоДиска`
- ✓ **Знаки пунктуации тоже не допускаются.** Здесь есть исключения, но вполне разумно в именах макросов знаки пунктуации не использовать вообще. Если в именах присутствуют знаки пунктуации. VBA выводит сообщение `Invalid procedure name` (Недопустимое имя процедуры).

Детальное обсуждение правил выбора имен в VBA приводится в главе 6.

Другие опции диалогового окна Запись макроса

В зависимости от приложения, диалоговое окно **Запись макроса** может содержать различные опции. Вот несколько примеров.

- ✓ Возможно, там будет место для ввода более подробного описания макроса.
- ✓ Возможно, у вас будут варианты при определении места хранения макроса. Как правило, предлагается хранить его либо в самом документе, либо в шаблоне документа.
- ✓ Возможно, вам предложат приписать макрос кнопке в панели инструментов либо назначить ему комбинацию клавиш. Если ваше приложение предусматривает такую опцию, используйте ее без сомнений! Подробности вы найдете в следующем разделе.

Кнопка, кнопка, ты чья?

Некоторые VBA-приложения позволяют приписать новый макрос кнопке в панели инструментов либо назначить ему комбинацию клавиш еще до начала записи этого макроса. Воспользуйтесь такой возможностью, ведь использовать кнопку или клавиатуру для вызова макроса удобнее всего.

Если приложение позволяет такой трюк, вы увидите в диалоговом окне Запись макроса соответствующее поле. Так, в Excel в диалоговом окне **Запись макроса** есть поле Сочетание клавиш, где вы можете указать комбинацию клавиш (например, <Ctrl+M>), которая должна будет запускать макрос (см. рис. 2.1). В Word диалоговое окно Запись

макроса содержит кнопки, которые позволят вам либо приписать макрос кнопке в панели инструментов, либо назначить ему комбинацию клавиш (рис. 2.2). Щелкнув на одной из этих кнопок, вы тем самым откроете новое диалоговое окно, в котором и должны будете сделать соответствующие назначения.

Когда вы назначаете комбинацию клавиш новому макросу, не поленитесь сразу же где-нибудь эту комбинацию записать. На время, пока эта новая комбинацию клавиш запомнится, держите напоминание о ней на своем рабочем столе или на экране компьютера.

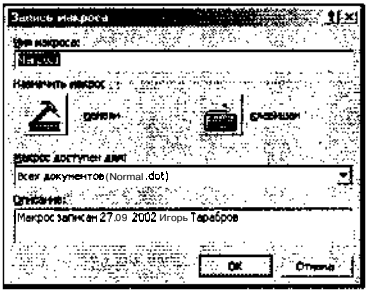


Рис. 2.2. Диалоговое окно Запись макроса в Word

Начало записи

Командой для начала записи макроса будет щелчок на кнопке OK в диалоговом окне **Запись** макроса. О том, что запись началась, вы узнаете по появившейся с этого момента панели инструментов Остановить запись (рис. 2.3). Изменится также и указатель мыши, превратившись в небольшое изображение магнитофонной компакт-кассеты.

Рис. 2.3. Панель инструментов Остановить запись

Запись команд

После начала записи вам не придется предпринимать никаких специальных действий, чтобы записать нужные вам команды в макрос. Нужно просто использовать команды приложения самым обычным способом. Все, что вы будете делать,— и выбор команд из меню, и форматирование изображений, и печатание текста — будет сохранено в макросе.

Ах, да! Все допущенные во время записи ошибки тоже будут добросовестно записаны и помещены в макрос. Чтобы их исправить, вы можете либо записать макрос вновь, либо отредактировать программный код макроса в редакторе Visual Basic (читайте ниже раздел "Редактирование программного кода макроса в редакторе Visual Basic").

Паузы во время записи и ее завершение

Чтобы остановить запись после выполнения всех команд, которые вы хотели записать, щелкните на кнопке Остановить **запись** в панели инструментов с тем же названием (см. рис. 2.3). Запись прекратится, а все записанные команды будут сохранены в виде VBA-программы. (Если вы

мне не верите, взгляните на полученный в результате программный код, — подробности ниже, в разделе "'Редактирование программного кода макроса в редакторе Visual Basic".)

Находясь в Word, вы можете сделать паузу в процессе записи макроса. Если вам необходимо выполнить команду, которая не должна быть частью макроса, щелкните на кнопке Пауза (тоже находится в панели инструментов Остановить запись). Процесс записи впадет в летаргический сон, с этого момента игнорируя все используемые вами команды.

Во время паузы в записи кнопка Пауза будет выглядеть нажатой (рис. 2.4). Официально теперь это будет кнопка Возобновить запись. Щелкните на этой кнопке, когда снова будете готовы продолжить запись временно покинутого вами макроса.

Рис. 2.4. Использование панели инструментов Остановить запись в Word

Выполнение макросов

Весь смысл записи макросов состоит в возможности их последующего воспроизведения или, если предпочитаете, *выполнения*. Наблюдение за тем, как достаточно сложный макрос выполняет за вас целую кучу утомительной работы, несомненно, можно отнести к тем маленьким радостям, которые делают жизнь приятной, (По причинам, которые я объясню немного позже, непременно сохраните свой документ перед тем, как вызвать для выполнения макрос, выполняющий значительные модификации содержимого документа.)

Ввиду того, что макросы являются VBA-программами, все приемы, которые используются при запуске созданных вручную VBA-программ, применимы и для автоматически записанных макросов. Всегда можно сначала открыть диалоговое окно Макрос (<Ait+F8>), выбрать в нем нужный макрос, а затем щелкнуть на кнопке Выполнить (в PowerPoint это кнопка Запуск; рис. 2.5). Подробно о технике выполнения VBA-программ говорится в главе 4.

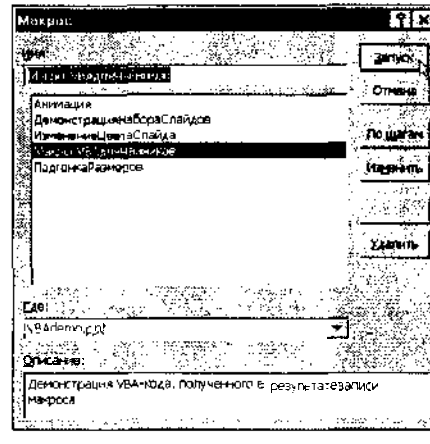


Рис. 2.5. Диалоговое окно Макрос (во многих VBA-приложениях оно называется именно так)

Автоматически записанный макрос не проявляет никакого "интеллекта" — макрос не может изменять свое поведение в зависимости от изменившихся условий. Будучи запущенным для выполнения, он пытается выполнить записанные команды, даже если в сложившейся ситуации в приложении эти команды вообще неприменимы. Например, если макрос предполагает, что вы находитесь в начале документа, а вы вызвали его, находясь в конце документа, вполне вероятно, что он не сделает ничего вообще. Если вызвать макрос, созданный для манипуляций с текстом, в то время, когда вы редактируете созданный вами рисунок, то такой макрос может свести на нет все ваши многочасовые сверхчеловеческие усилия по созданию шедевра, просто уничтожив последний. Мой вам совет: независимо от сложности макроса, сохраните документ перед тем, как выполнять этот макрос, и будьте всегда готовы воспользоваться командой Отменить (<Ctrl+Z>).

Редактирование макросов как способ создания программного кода

Не следует думать, что средству записи макросов останется только собирать электронную пыль, когда вы научитесь самостоятельно создавать VBA-код. В действительности, тогда вы сможете использовать это средство еще чаше. В конце концов, наиболее эффектные VBA-программы сильно зависят от объектов, методов и свойств, заимствованных именно из приложения. Вместо того чтобы печатать соответствующие команды вручную, почему бы не записать их как макрос?

Не забывайте, что в результате записи макроса все использованные вами команды будут записаны в виде последовательности соответствующих операторов языка VBA. Полученный таким образом программный код можно использовать в качестве отправной точки при создании своей программы, добавляя в него новые операторы по мере необходимости. По крайней мере, для команд, записанных как макрос, вам не придется вспоминать точную форму входящих в них операторов VBA и заботиться о допущенных при наборе ошибках.

Кроме того, такой подход предлагает самый простой способ связать с кнопками в панелях инструментов и с комбинациями клавиш свои собственные программы, конечно, если приложение позволяет сделать соответствующие назначения при записи макроса {см. выше раздел "Кнопка, кнопка, ты чья?").

Редактирование программного кода макроса в редакторе Visual Basic

После того как макрос записан, полученную VBA-программу вы можете отредактировать (или, по крайней мере, просто взглянуть на строки ее программного кода). Вот как это сделать.

1. Выберите **Сервис⇨Макрос⇨Макросы** или нажмите <Alt+F8>, чтобы открыть диалоговое окно **Макрос** (см. рис. 2.5).
2. Выберите нужный вам макрос из списка ниже поля **Имя.**
   Макрос, который вы только что записали, не выбирается автоматически, поэтому, чтобы найти его, вам, вероятно, придется полистать список.
3. Щелкните на кнопке **Изменить.**

Перед вами немедленно возникнет окно редактора Visual Basic с помещенным в него программным кодом выбранного макроса, готовым для редактирования. Пример показан на рис. 2.6.

Простейшие усовершенствования макросов

В этой главе не предполагается подробно рассматривать приемы редактирования макросов в редакторе Visual Basic. В конце концов, это те же самые приемы, которые используются при создании любых VBA-программ и которые, в основном, обсуждаются в этой книге.

Все же я считаю целесообразным предложить здесь несколько простых и обычно полезных способов усовершенствования макросов. Предлагаемые идеи требуют минимальных программистских усилий, но позволяют сделать макрос более гибким. Попробуйте следующее.

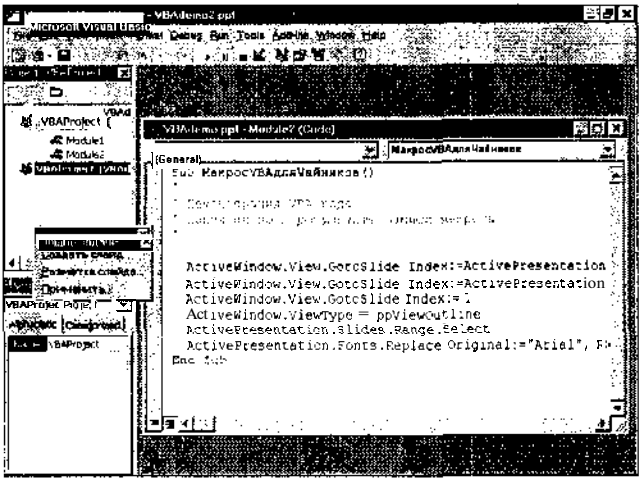


Рис. 2.6. Запись макроса завершена, и перед вами - его программный код, созданный генератором средства записи макросов

- ✓ **Организуйте повторение команд в макросе.** Возможно, вам известно наперед, что придется выполнить целую последовательность команд (или какую-то одну команду) ровно 11 раз. В этом случае все, что вам нужно, — это добавить в свой программный код цикл `For...Next`. Вы спрашиваете, что такое цикл `For...Next`? Это один из элементов языка программирования VBA, обсуждать которые я собираюсь в главе 8.
- ✓ **Организуйте повторение команд в макросе с выяснением количества необходимых повторений непосредственно во время работы макроса.** Если вы достаточно амбициозны, добавьте в свой макрос сразу и цикл `For...Next`, и функцию `InputBox`. Последняя выводит на экран диалоговое окно, предлагающее ответить на некоторый вопрос, в данном случае — о количестве повторений. Использование `InputBox` обсуждается в главе 11.
- ✓ **Добавьте при вызове возможность выбора текста для использования макросом.** Макрос, который всегда вставляет в документ один и тот же текст, не слишком гибок. Добавив в записанный макрос функцию `InputBox`, вы сможете сообщать при каждом запуске макроса, какой текст нужно вставить. Снова напомню, что подробности об использовании `InputBox` вы найдете в главе 11.
- ✓ **Добавьте возможность выбора перед запуском макроса.** Довольно часто для правильной работы макроса требуется выбрать текстовый, графический или какой-то другой объект. Структура `If...End If` языка VBA позволяет проверить, выбран

ли объект подходящего типа, чтобы иметь возможность отменить работу макроса в случае невыполнения каких-либо условий. Все тонкости использования оператора `If...End If` обсуждаются в главе 8.

Как только вы осознаете привлекательность использования средства записи макросов в совокупности с самыми минимальными усилиями по добавлению VBA-кода, вас непременно привлечет и программирование. После этого ваше превращение в компетентного VBA-программиста станет только вопросом времени и не потребует слишком больших усилий.

Глава 3

Основы программирования на VBA

В этой главе...

- Запуск редактора Visual Basic
- Использование справочной системы VBA
- Программирование в VBA — краткое и нестрогое руководство

Макросы хороши до определенных пределов, но гораздо больше можно получить от полноценных VBA-программ. Эта глава послужит аварийным введением в рациональное VBA-программирование. После вводного обзора редактора Visual Basic и справочной системы VBA мы с вами на простом примере разберем по шагам процесс построения VBA-программы. Эти знания станут тем солидным базисом, который потребуется для освоения всего остального материала книги.

Вызов редактора Visual Basic

Редактор Visual Basic служит командным центром для работы в VBA. В нем вы должны находиться при разработке VBA-форм, создании VBA-кода, тестировании и отладке VBA-программ. Экспертом по использованию редактора Visual Basic вы станете после прочтения главы 5, а пока вам нужно только знать, как вытянуть этот редактор на экран.

Если вы делали кое-что из того, о чем так много говорилось в предыдущей главе при обсуждении программного кода, создаваемого средством записи макросов, вы должны уже знать один из способов вызова редактора Visual Basic — выбор макроса в диалоговом окне Макрос с последующим щелчком на кнопке Изменить. Но, конечно же, можно запустить редактор Visual Basic и непосредственно. В большинстве VBA-приложений можно воспользоваться одним из следующих методов.

- ✓ Выбрать из меню Сервис⇨Макрос⇨Редактор Visual Basic.
- ✓ Нажать <Alt+F11>. Вы услышите, как затараторит жесткий диск, и через несколько мгновений на экране появится редактор Visual Basic. Он должен хотя бы отдаленно напоминать то, что изображено на рис. 3.1.

В некоторых приложениях вам понадобится проделать другой маршрут, чтобы добраться до редактора Visual Basic. Однако вы в любом случае найдете необходимую команду где-нибудь в меню Tools (Сервис).

Вызов редактора Visual Basic одним щелчком

Если в вашем приложении есть кнопка в панели инструментов для вызова редактора Visual Basic, используйте ее. В VBA-приложениях из пакета Microsoft Office (Word, Excel и PowerPoint) эта кнопка помещена в панель инструментов Visual Basic (рис. 3.2). Наборы кнопок в этой панели инструментов у разных приложений могут немного отличаться.

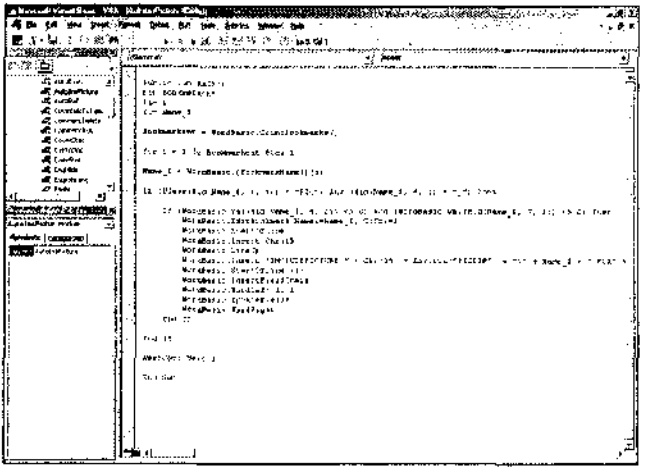
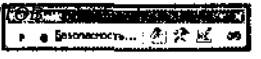


Рис. 3.1. Редактор Visual Basic

Рис, 3.2. Вид панели инструмен
Visual Basic в приложении Word



Кнопки в этой панели инструментов, как правило, соответствуют пунктам подменю Макрос из меню Сервис. Вот для чего предназначены некоторые из этих кнопок (слева направо, см. рис. 3.2).

- ✓ Первые две кнопки предназначены соответственно для выполнения и записи макросов.
- ✓ Кнопка справа от кнопки Безопасность вызывает редактор Visual Basic. Если вы часто используете редактор Visual Basic, а другие кнопки в этой панели инструментов вам не нужны, скопируйте кнопку вызова редактора в другую панель. Чтобы скопировать кнопку в любом из приложений Office, перетащите ее на новое место, удерживая при этом нажатой клавишу <Alt>.
- ✓ Кнопка с изображенными на ней молотком и гаечным ключом открывает панель элементов управления, очень похожую на панель элементов управления VBA. В Office вы можете поместить элементы управления VBA в сам документ, а не только в VBA-форму.
- ✓ Кнопка с изображенными на ней линейкой, треугольником и карандашом включает режим проектирования, в котором вы можете редактировать элементы управления VBA в своем документе.

Краткое знакомство с редактором Visual Basic

Первые обращения к редактору Visual Basic могут вас озадачить. Наверное, присутствие меню и панелей инструментов вверху экрана покажется вам привычным, но вот что означает это обилие окон? Предсказать их взаимное расположение практически невозможно, поскольку они размещаются и комбинируются почти в бесконечном разнообразии вариантов (даже когда вы открываете редактор Visual Basic впервые, все равно предстающая перед вами картина допускает немало вариантов, в зависимости от того, как вы его открыли). Кроме того, если вы не профессиональный программист, названия окон и их функции окажутся для вас наверняка непонятными.

Научить вас уверенно обращаться со всеми этими окнами должна глава 5, здесь же я приведу только список их имен и функций (табл. 3.1).

Таблица 3.1. Имена и функции

Имя окна	Функциональное назначение
Project Explorer (окно проводника проекта)	Перемещение по компонентам VBA-проекта и управление ими
Code (окно программного кода)	Просмотр, добавление и редактирование программного кода VBA
UserForm (окно формы)	Проектирование пользовательских форм (диалоговых и других окон)
Toolbox (панель элементов управления)	Добавление элементов управления (текстовых полей или кнопок} в формы, а во многих VBA-приложениях и в документы
Properties (окно свойств)	Установка индивидуальных атрибутов выделенной формы или элемента управления
Watch (окно контролируемых выражений)	Отслеживание значений выбранных переменных программы и выражений
Locals (окно локальных переменных)	Отслеживание значений переменных текущей процедуры
Immediate (окно немедленного выполнения)	Выполнение отдельных строк программного кода для немедленного получения результата
Object Browser (окно обозревателя объектов)	Исследование объектов, доступных программам

Давайте поближе рассмотрим окно программного кола, предназначенное для печати и редактирования в нем VBA-кода, и окно пользовательской формы, предназначенное для визуального проектирования окон и форм. Примеры окон программного кода и пользовательской формы показаны на рис. 3.3 и 3.4 соответственно.

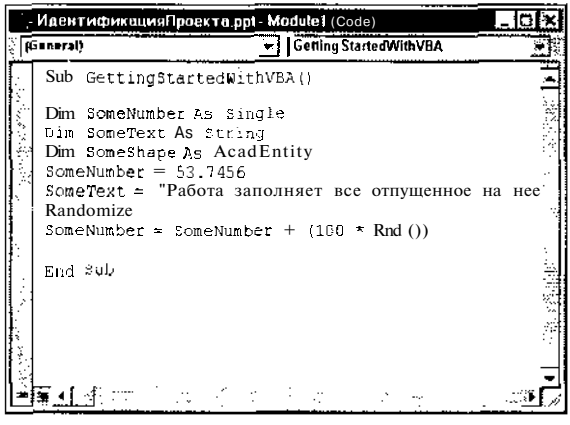


Рис. 3.3. В окнах программного кода, подобных этому, печатается программный код VBA

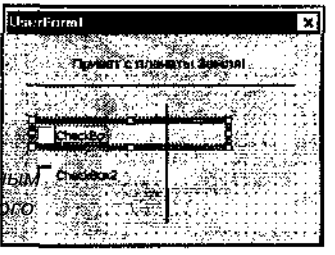


Рис. 3.4. Пример окна пользовательской формы с помещенным внего диалоговым окном на стадии еще незавершенного проектирования

С самого начала перед вами откроется несколько окон программного кода и окон пользовательской формы, а может случиться так, что их не будет на экране вообще. Но для нас это сейчас не имеет значения, поскольку после краткого обсуждения справочной систе-мы VBA нам с вами понадобятся новые окна для программы-примера, которую вы почти готовы создать.

На помощь!

Практически все современные приложения до отказа забиты говорящими сами за себя пунктами меню, кнопками в панелях инструментов с красивыми маленькими пиктограммами, ясно соответствующими выполняемым кнопками командам, а также всплывающими сообщениями-подсказками, которые объясняют функциональное назначение этих кнопок. Кому еще при этом требуется какая-то справочная система? Например, вам, если вы собираетесь создавать VBA-программы.

Для начала вам потребуется справка о самом редакторе Visual Basic. Хотя меню и панели инструментов этого редактора работают точно так же, как и во всех остальных приложениях, поначалу многие из предлагаемых редактором команд выглядят совершенно непонятными. Кроме того, редактор Visual Basic озадачивает вас сразу восемью-девятью типами окон, и требуется немало времени, чтобы разобраться с ними.

Ясно, что с пользовательским интерфейсом редактора Visual Basic вы разберетесь довольно скоро. Но даже самым опытным программистам требуется помощь, когда дело касается деталей самого языка VBA. Из-за сотен объектов, методов, ключевых слов и Бог знает, чего еще. что просто нельзя удержать в голове, без файлов справки VBA работа становится практически невозможной.

Существует созданная *Microsoft* "бумажная" версия руководства по программированию в VBA, но комплект поставки далеко не всякого VBA-приложения предполагает наличие печатной копии. Определенно можно сказать, что такая копия руководства *есть* в комплекте пакета Office для разработчика.

Скорая помощь VBA

Окна справки VBA выглядят по-разному в различных VBA-приложениях. Справочная система и старых программах использует старое и проверенное ядро WinHelp, своими истоками восходящее еще к Windows 3.11. Ядром же справки VBA 6 является HTML Help — новый стандарт *Microsoft.* Впервые представленная в Windows 98, система HTML Help больше похожа на Web-броузер. К сожалению, те версии HTML Help, которые я использовал до настоящего времени, ведут себя временами довольно странно и непредсказуемо, а кроме того, в них отсутствуют некоторые полезные возможности WinHelp, например возможность удерживать окно справки на экране перед другими окнами, когда активно другое приложение.

Чтобы открыть главное окно справки VBA, выберите Help⇨Справка по **Microsoft Visual** Basic — появится окно справки с описанием VBA в общем. Окно HTML Help для VBA 6 показано на рис. 3.5. К сожалению, здесь вы не найдете справки о работе с конкретным приложением.

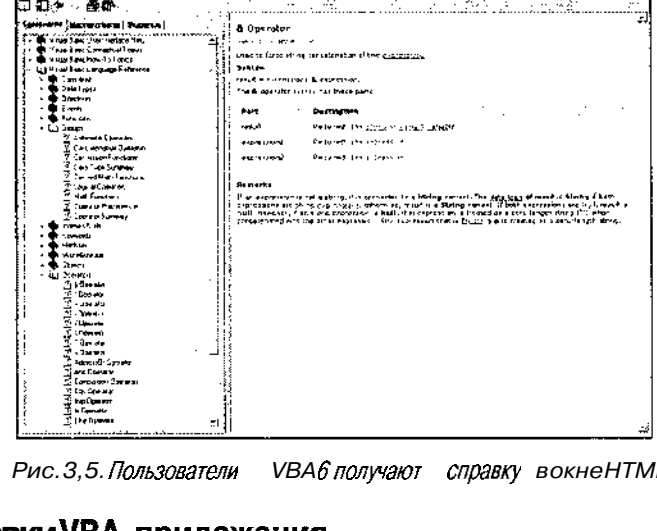


Рис. 3,5. Пользователи VBA6 получают справку вокне HTML Help

Вызов справки VBA-приложения

Печально, но факт — редактор Visual Basic не предлагает команду, обеспечивающую непосредственный доступ к файлам справки того VBA-приложения. из которого он вызван. Не тратьте время на поиски — такой команды просто нет. Если вам понадобится дополнительная информация о программировании в данном приложении, а не о VBA вообще, единственным решением будет следующий четырехшаговый процесс.

1. **Откройте обозреватель объектов, нажав <F2> или выбрав Вид⇨Обозреватель объектов (подробно обозреватель объектов рассмотрен в главе 6).**
2. **Из раскрывающегося списка в верхнем левом углу окна обозревателя объектов выберите имя вашего VBA-приложения.**
3. **Выделите любой раздел в панели классов в левой части окна обозревателя объектов.**
4. **Нажмите <F1> или щелкните на кнопке с вопросительным знаком вверху окна обозревателя объектов, чтобы вызвать справочную систему.**

   При этом вы, конечно, увидите ту часть справки, которая относится к выделенному вами разделу в окне обозревателя объектов, но зато в окне справки можно воспользоваться вкладкой Содержание, на которой представлена вся справочная система вашего приложения.

Поиск в стоге справок

Работа справочной системы VBA организована в соответствии со стандартами Windows как в WinHelp, так и в HTML Help. В обоих случаях вы можете сделать следующее.

- ✓ Воспользоваться вкладкой **Содержание,** чтобы увидеть дерево разделов справки, подобное содержанию обычной книги.

- ✓ Воспользоваться вкладкой Предметный указатель, чтобы увидеть алфавитный список ключевых слов, предлагаемый создателями справки.

Можно также воспользоваться вкладкой Поиск, чтобы найти практически любое слово, встречающееся в файлах соответствующей справки. Следует, правда, заметить, что в справке VBA для Office вместо вкладки Поиск есть вкладка Мастер ответов. Предполагается, что вы можете теперь перейти к нужному разделу справки в ответ на целый введенный вами вопрос, но с тем же успехом вы можете просто ввести одно-два слова, а затем щелкнуть на кнопке Найти.

Контекстно-зависимая справка

В редакторе Visual Basic клавиша <F1> вызывает *контекстно-зависимую* справку, т.е. открывает перед вами именно тот раздел справки, который соответствует работе, выполняемой вами в момент вызова. То, что вы увидите, зависит от типа окна, которое будет в редакторе при этом активным, а часто и от того, что именно вы делаете в этом окне.

Такая возможность оказывается действительно удобной при создании программного кода и проектировании форм. В окне программного кода клавиша <F1> предоставит справку для термина, находящегося в точке ввода (точка ввода задается текстовым курсором — мигающей вертикальной линией, определяющей место появления печатаемых вами символов). Не волнуйтесь, если вы забыли, как используется какой-нибудь VBA-оператор. например For . . . Next, или если не помните, какими свойствами и методами обладает нужный вам объект. Просто поместите текстовый курсор в соответствующий оператор и нажмите <F1>. Пример того, что вы можете при этом увидеть, показан на рис. 3.6.

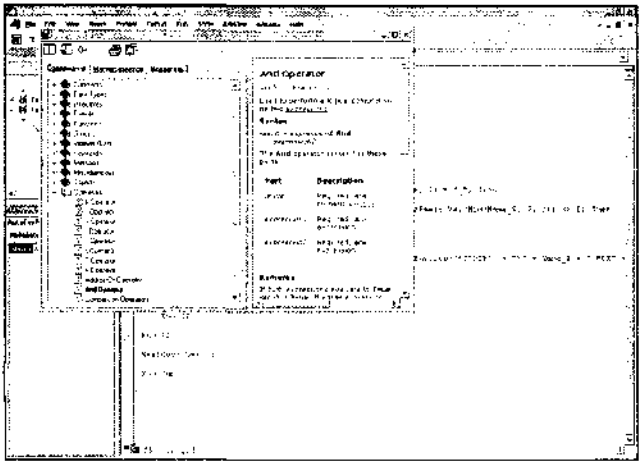


Рис. 3.6. Получение справки для метода Add в Visio. Если присмотреться, можно заметить, что точка ввода в окне программного кода находится внутри ключевого слова Add

Подобным образом клавиша <F1> предоставит вам самую подробную справку о проектировании форм. В окне пользовательской формы можно получить справку о каждом из поме-

щенных в форму элементов управления (рискуя показаться назойливым, я напомню, что *элементы управления* — это кнопки, флажки, фреймы и т.п.). Щелкните на элементе управления, чтобы выделить его, а затем нажмите <F1>. Чтобы получить справку об определенном *свойстве* элемента управления, выделите этот элемент управления, щелкните в строке соответствующего свойства в окне свойств, а затем нажмите <F1>.

В других окнах клавиша <F1> работает следующим образом.

- ✓ В окне проводника проекта и окне пользовательской формы нажатие клавиши <F1> вызывает раздел справки с описанием окна. Для окна пользовательской формы это сработает, только если не выделен ни один из элементов управления.
- ✓ В окне контролируемых выражений, окне локальных переменных и в окне немедленного выполнения нажатие <F1> вызывает окно справки со множеством предлагаемых в нем *ссылок* — подчеркнутых слов и словосочетаний, на которых надо щелкнуть, чтобы перейти к соответствующему разделу справки. Лично мне совершенно не понятно, почему сделано так, что для этих окон подходящий раздел справки не появляется автоматически.
- ✓ В окне обозревателя объектов при выделенных классе и члене класса нажатие <F1> вызывает появление раздела справки, соответствующего выбранному объекту.

Что особенного в разделах справки VBA

При использовании справки VBA обращайте внимание на ссылки и кнопки ссылок, вызывающие другие относящиеся к рассматриваемому вопросу разделы справки. Научитесь пользоваться ими! Подобно разделу, показанному на рис. 3.7, большинство разделов справки VBA предлагают такую возможность. Если показанный раздел не содержит необходимой вам информации, вполне вероятно, что вы отыщете ее в разделах, предлагаемых ссылками.

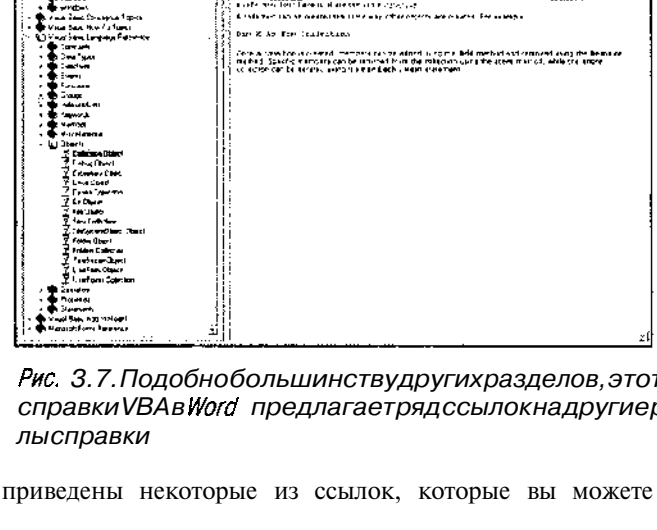


Рис. 3.7. Подобно большинству других разделов, этот раздел справки VBA в Word предлагает ряд ссылок на другие разделы справки

В табл. 3.2 приведены некоторые из ссылок, которые вы можете увидеть в окне справки.

Таблица 3.2. Ссылки и кнопки ссылок

Ссылка или кнопка	Назначение
See also (См. также)	Щелкните на этой ссылке, чтобы увидеть информацию о других разделах справки, относящихся к рассматриваемому вопросу
Example (Пример)	Увидев кнопку или ссылку Example, без сомнения используйте ее. Примеры существенно помогают в понимании порой запутанных и неполных объяснений в файле справки. Кроме того, вы можете скопировать пример в свою VBA-программу и подправить его по своему усмотрению (врезка "Позаимствуйте программный код из файла справки")
Methods, Properties, Events (Методы, свойства и события)	Эти ссылки имеются в разделах справки, описывающих объекты. Позволяют ознакомиться с методами, свойствами и событиями, которые можно использовать с данным объектом
Applies to (Применяется к...)	Эта ссылка присутствует тогда, когда раздел справки относится к свойству, методу или событию. Щелкните на ней, чтобы увидеть список объектов, с которыми свойство, метод или событие можно использовать

Позаимствуйте программный код из файла справки

Не упустите возможность использовать фрагменты программного кода, предлагаемые во многих разделах справки VBA. Нередко программный код предлагаемого примера после самых минимальных изменений будет в точности соответствовать тому, что требуется вам. Чтобы использовать примеры справки, выделите нужные вам строки программного кода с помощью мыши, щелкните на выделенном правой кнопкой мыши и в появившемся меню выберите Копировать. После этого перейдите в окно программного кода в редакторе Visual Basic и щелкните на кнопке Вставить в панели инструментов или нажмите <Ctrl+V>.

Установка всех файлов справки

В зависимости от приложения и от того, как вы его инсталлировали, на вашем жестком диске установлен, а, может быть, и не установлен полный комплект файлов справки VBA. Из-за того, что любая серьезная работа в VBA без файлов справки практически невозможна, убедитесь в том, что все они установлены.

Например, программа установки Office устанавливает не все, а только некоторые файлы справочной системы VBA. Большинство пользователей при установке Office на свои компьютеры используют в программе установки кнопку Типичная. Это, конечно, очень удобно (зачем думать?), но при этом не устанавливается целый ряд важных файлов справки VBA. Если так же действовали и вы, снова запустите программу установки и выберите опцию **Выборочная.** По очереди для каждого из приложений Office выберите пункты **Справка** и **Файлы примеров.** Когда вы затем щелкнете на кнопке **Применить,** появится диалоговое окно, подобное показанному на рис. 3.8. В этом окне отметьте флажок пункта **Справка** по Visual Basic.

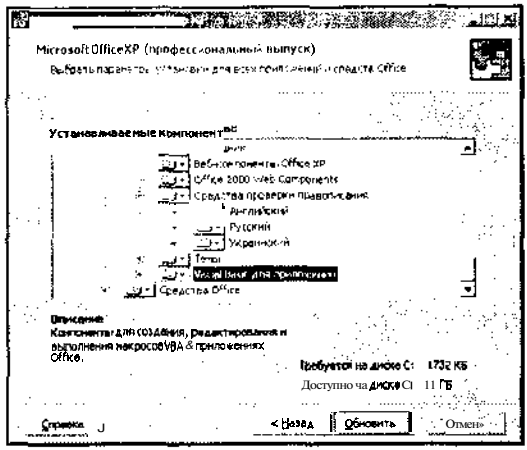


Рис. 3.8 Чтобы установить справку VBA, отметьте флажок Справка для Visual Basic

В Office, даже если вы использовали при установке опции **Выборочная** или **Полная,** файлы справки VBA для Outlook все равно не устанавливаются на ваш жесткий диск. Чтобы установить справку VBA для Outlook, вам придется самим скопировать соответствующий файл справки из папки `valupack\morehelp` на компакт-диске Office в папку, куда установлен Office.

Создание VBA-программы

Теперь, после знакомства с редактором Visual Basic и системой справки VBA, вы полностью готовы к своему первому походу к границам программирования. Я уже упоминал в главе 1, что процесс создания программы можно разделить на несколько этапов. В такой изысканной книге, как эта, я просто не могу себе позволить пожертвовать столь прекрасно упорядоченным подходом. Так что вам придется пройти весь курс строевой подготовки.

1-й шаг. Проектирование программы-примера

Поскольку мне не известно, с каким из VBA-приложений вы работаете, придется предложить вам в качестве примера нечто вполне типичное и, с другой стороны, потенциально полезное. Наша программа-пример должна будет открывать на экране новое окно с показанной в нем злободневной цитатой, а также датой и временем (я называю все это *сообщением).* Окно будет оставаться на экране до тех пор, пока пользователь не щелкнет на кнопке OK.

Имея такое описание программы, вы можете без труда представить себе те элементы, из которых должна состоять программа.

- ✓ Очевидно, программа имеет одно окно, поэтому вам понадобится одна форма (UserForm).
- ✓ Для формы потребуются два элемента управления — надпись для сообщения и кнопка для команды OK.
- ✓ Нужно будет также создать программный код для двух процедур: одной — для надписи, в которую нужно поместить сообщение, а другой — для выхода из программы, когда пользователь щелкнет на кнопке OK.

Чего наша программа определенно *не* требует, так это отдельного модуля для ее VBA-кода *(модуль* — это отдельная единица программного кода, содержащая одну или несколько процедур; более подробно о модулях говорится в главе 6). Обе наши процедуры можно разместить в окне программного кода, ассоциированном с формой. В конце концов, ведь эти процедуры отвечают на *события,* которые происходят с формой.

Рассмотрим процедуру, отображающую сообщение и дату. Нам она не понадобится до тех пор, пока окно программы не появится на экране. В программе это произойдет, конечно, быстрее, чем вы глазом успеете моргнуть, но принцип остается — все равно эта процедура должна вызываться событием появления формы.

Процедура выхода из программы тоже вызывается событием, относящимся к форме, а именно — щелчком на кнопке OK. Эту процедуру тоже можно разместить в окне программного кода формы.

Не забывайте, что у многих VBA-программ вообще нет своих окон. При создании такой программы вам непременно придется создать хотя бы один модуль, чтобы было куда поместить программный код.

2-й шаг. Реализация проекта

Теперь, когда план составлен, вы можете со всей серьезностью приступить к программированию. Возьмите в качестве отправной точки разработку внешнего вида формы, дополните ее небольшим, но тщательно выверенным фрагментом программного кода, и вы вправе надеяться в результате на небольшой VBA-шедевр.

Добавление новой формы

Первым делом в редакторе Visual Basic создайте новую, чистенькую форму, которая послужит окном программы. Из меню редактора Visual Basic выберите Insert⇨UserForm, чтобы поместить новую форму на свой экран.

Видите, как легко это делается? На заре программирования для Windows приходилось вручную выписывать длинные и довольно сложные операторы даже для таких простых вещей, как помещение на экран обычного чистого окна.

Как видно из рис. 3.9, новая пользовательская форма представляет собой невыразительную серую панель. Обратите внимание, вы можете изменять размеры формы, перетаскивая маленькие белые квадратики, примыкающие к правому и нижнему краям формы (эти квадратики

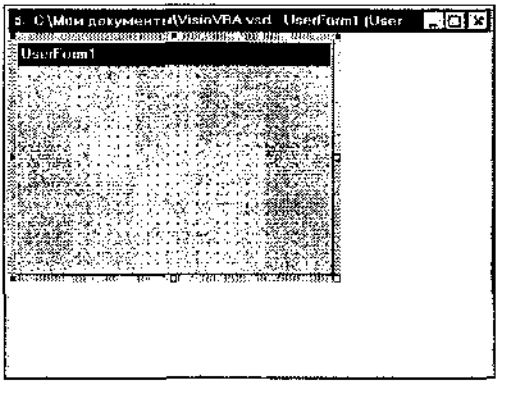


Рис. 3.9. Только что созданная форма в поисках цели своего существования

называются *маркерами* изменения размеров). Окно для нашей программы должно быть немного шире (чтобы в нем поместилось все сообщение одной строкой) и немного ниже (одна строка текста и одна кнопка не займут много места).

Значительно интереснее будет панель элементов управления (рис. 3.10), представляющая собой небольшое окно с набором пиктограмм, которое возникает на экране при работе с формой.

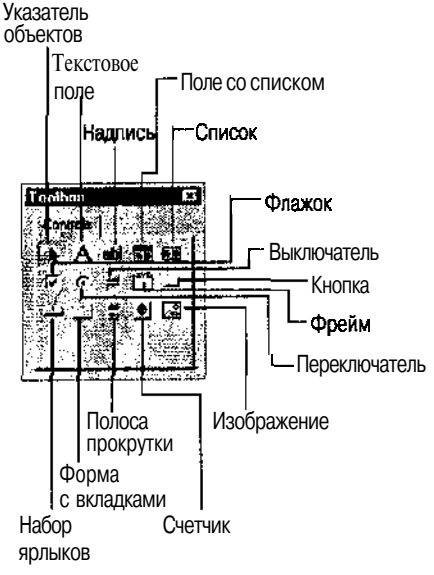


Рис. 3.10. Панель элементов управления VBA с пиктограммами элементов управления, которые вы можете поместить в свою форму

Добавление надписи в форму

Теперь вы можете несколько оживить форму, добавив в нее пару элементов управления. Начнем, пожалуй, с *элемента управления надписью,* который просто отображает текст. Во время выполнения программы элемент управления надписью отображает текст, который пользователь видит, но изменить не может.

Но ваша программа это может. Нашей с вами программе-примеру это просто необходимо, так как ей придется отображать дату и время, которые зависят от того, когда выполняется программа.

Чтобы поместить надпись в свою новую пользовательскую форму, выполните следующее.

1. Убедитесь, что форма активна, щелкнув на ней. Панель элементов управления видна только тогда, когда форма активна.
2. Щелкните на пиктограмме с буквой А в панели элементов управления.
3. Поместите указатель мыши в форму, ближе к левому верхнему краю, и, нажав левую кнопку мыши, перетащите указатель вправо вниз, чтобы появившийся при этом прямоугольник вместил нужное сообщение (используйте рис. 3.11 в качестве руководства).

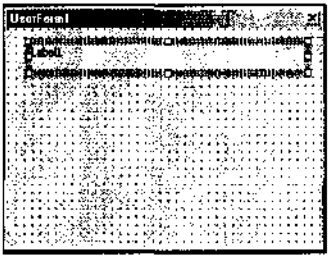


Рис. 3.11. После помещения в форму элемента управления надпись она должна выглядеть приблизительно так

Каждой новой надписи VBA автоматически приписывает заголовок, поэтому в прямоугольной рамке, ограничивающей наш элемент управления надписью, вы увидите величественный текст Label1. Ясно, что это не тот текст, который нам нужен. Чтобы удалить автоматически созданный текст надписи, придется обратиться у окну свойств.

Использование окна свойств для настройки элементов управления

Каждый элемент управления имеет довольно длинный список *свойств*. Они определяют внешний вид и поведение элемента управления при выполнении программы. Одним из важнейших достоинств VBA является возможность изменять свойства элементов управления без печатания программного кода.

Для управления свойствами предназначено окно свойств. Как видно из рис. 3.12, в окне свойств представлен список всех свойств выделенного в данный момент элемента управления. Чтобы изменить нужное вам свойство, просто найдите его в левом столбце и измените значение этого свойства в правом.

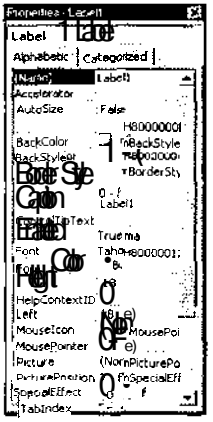


Рис. 3.12. Для изменения установок любого из свойств элемента управления используйте окно свойств

Для надписи в нашей программе-примере нужно изменить лишь два свойства — Name (Имя) и Caption (Заголовок). Замените имя элемента управления с Label1 на lblNow (первые три буквы указывают на то, что имя принадлежит элементу управления надписью). Чтобы сделать это, найдите свойство Name в самом верху списка свойств (оно представлено как (Name)), — это единственное из свойств, заключенное в скобки. Двойным щелчком в правом столбце строки этого свойства выберите его, а затем впечатайте новое имя. (Имена элементов управления используются при создании программного кода, поэтому для них имеет смысл выбирать имена, которые напоминали бы о назначении элементов управления. В главе 6 говорится о соглашениях, которые следует использовать в именах для идентификации создаваемых элементов управления.)

Затем опуститесь по списку ниже и найдите свойство Caption. Здесь просто удалите значение свойства. Почему? Просто потому, что ваша программа сама обеспечит текст для надписи во время выполнения.

Теперь ясно, что вам придется создавать некий программный код, если вы хотите, чтобы программа во время выполнения изменяла свойства элемента управления. Большинство свойств при этом остаются неизменными, а поэтому создание такого программного кода не требует больших усилий. Раз уж вы меняете свойства, заодно измените и заголовок самой формы. Выберите форму, щелкнув на полосе заголовка окна формы. В изменившемся при этом окне свойств найдите

свойство Caption и измените его на что-нибудь типа Вы не видели моей клавиатуры? или Моя первая программа в VBA. Этот новый заголовок появится в полосе заголовка формы.

Добавление кнопки команды

Элемент управления кнопкой команды (чаще называемый просто кнопкой) отличается от надписи тем, что предполагает определенные действия со стороны пользователя программы. Когда кто-нибудь щелкает на кнопке, ее изображение на экране изменяется так, что она выглядит нажатой, а программа в ответ выполняет какие-нибудь действия.

Нашей программе нужна всего одна кнопка, которая после щелчка на ней завершит выполнение программы. Чтобы поместить такую кнопку в форму, выполните следующее.

1. **Щелкните в окне формы, чтобы сделать его активным снова.**
   При этом панель элементов управления, которая исчезла, когда вы перешли в окно свойств, возникнет снова.
2. **Щелкните на той пиктограмме в панели элементов управления, на которой изображена кнопка (см. рис. 3.11).**
3. **Начав немного левее середины формы и нажав левую кнопку мыши, перетащите указатель по диагонали вниз, чтобы создать кнопку.**

Теперь форма должна принять вид. как на рис. 3.13.

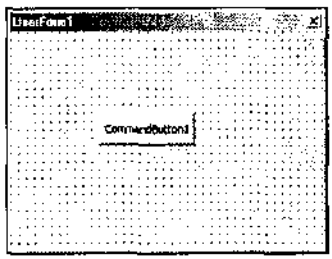


Рис. 3.13. ВИДформыпослетого, каквнеедобавиликнопку

Теперь вы должны установить нужные значения свойств кнопки. Используя окно свойств, выполните следующие шаги.

1. **Измените значение свойства (Name) на `OKButton`.**
   Напомню снова, что, поскольку при создании программного кода вам придется ссылаться на кнопку по ее имени, такое имя кнопки напомнит вам о ее назначении.
2. **Измените значение свойства Caption на OK. Этот текст увидит пользователь на кнопке во время выполнения вашей программы.**

Вот вы и создали форму для своей программы. Уже к этому моменту ваша программа стала вполне функционирующей VBA-программой — она будет показывать очень симпатичное только что созданное вами окно. Чтобы запустить программу на выполнение, нажмите <F5> или щелкните на кнопке Выполнить в панели инструментов редактора Visual Basic. На этой кнопке изображена маленькая направленная вправо стрелка.

Когда вы запустите эту наполовину испеченную программу, на экране появится форма на фоне вашего VBA-приложепня (а не на фоне окна редактора Visual Basic;

рис. 3.14). Поле надписи будет пустым, и кнопка OK ничего делать не будет —- не забывайте, что никакого программного кола мы еще не печатали. Чтобы завершить выполнение программы, щелкните на стандартной для Windows кнопке закрытия в правом конце строки заголовка окна.

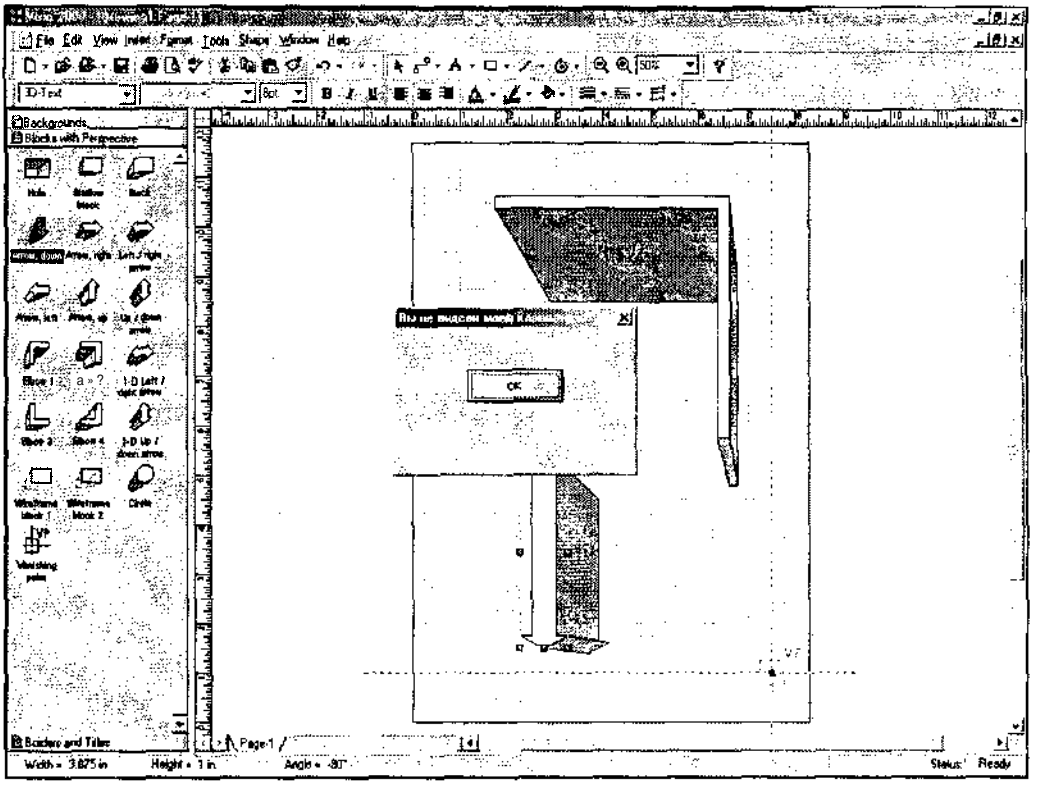


Рис. 3.14. Запуск пользовательской формы перед добавлением программного кода

Добавление программного кода

Пора перейти к наиболее пугающему и вместе с тем самому увлекательному этапу программирования в VBA — созданию программного кода. Напомню, что для нашей с вами программы требуется создать две процедуры и что они связаны с событиями, возникающими в процессе выполнения программы. Первая процедура должна при появлении на экране формы отобразить нужное сообщение, а вторая — завершить выполнение программы, когда кто-нибудь щелкнет на кнопке OK.

Вызов окна программного кода формы

Чтобы напечатать связанный с формой или одним из ее элементов управления программный код, вам нужно вместо окна формы открыть окно программного кода этой формы. Для этого выделите форму или элемент управления в ней. Для кнопки OK это сделать легче всего, так почему бы не начать именно с нее? Щелкните на кнопке OK, чтобы вокруг нее появились маркеры изменения размеров. Теперь для вызова окна программного кода воспользуйтесь любым из следующих способов.

- ✓ Выбрать View⇨Code из меню.
- ✓ Нажать <F7>.
- ✓ Щелкнуть на форме правой кнопкой мыши и в появившемся контекстном меню выбрать View **Code.**

В появившемся при этом окне программного кода должна уже быть заготовка процедуры (рис. 3.15). VBA автоматически создает процедуру для часто используемого в случае кнопки события — простого щелчка на кнопке. Ваша программа использует эту процедуру, когда кто-нибудь щелкнет на кнопке OK.

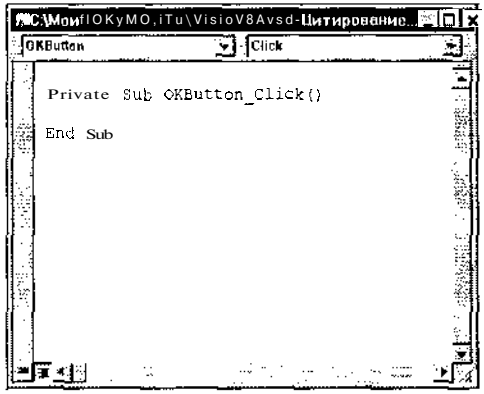


Рис. 3.15. VBA создаст такую заготовку процедуры, как только вы откроете окно программного кода для кнопки OK

По сути, эти две строки программного кода не делают ничего — они просто определяют рамки, показывающие VBA, где начинается и заканчивается процедура. Первой строкой созданного для вас программного кода будет

```
Private Sub OKButton_Click()
```

В любой VBA-процедуре первая строка программного кода определяет тип процедуры (в данном случае это процедура типа **Sub,** т.е. подпрограмма) и имя процедуры. Private и Sub относятся к *ключевым словам* VBA, т.е. к словам и символам, которые являются частью языка VBA. Ключевые слова имеют фиксированные специальные значения в VBA, и вы не можете их использовать в качестве имен таких объектов, как процедуры. Кстати, различные виды процедур в VBA обсуждаются в главе 6. В данном случае VBA предлагает для процедуры имя OKButton_Click, которое представляет собой комбинацию имени кнопки и типа события.

Последней строкой автоматически генерируемого программного кода будет

```
End Sub
```

Такой строкой должны заканчиваться все процедуры типа Sub. Эта строка сообщает VBA о том. что выполнение процедуры пора завершить.

Добавление своего программного кода

Текстовый курсор будет мигать между этими двумя созданными VBA строками. Напечатав лишь одну дополнительную строку программного кода, мы дадим возможность программе завершить работу. Вот этот программный код:

```
Unload Me
```

Оператор Unload убирает указанный объект из памяти. Здесь это объект с именем Me, имеющим в VBA специальный смысл. В данном случае оно означает форму и весь ее программный код.

Исправление ошибок при вводе программного кода

Фигурально выражаясь, всегда, когда вы печатаете программный код, за вашей спиной стоит компилятор VBA. Если компилятор обнаруживает очевидную ошибку, вы немедленно получаете соответствующее сообщение с информацией об ошибке, по крайней мере в общих словах. Предположим, например, что после оператора Unload Me вы напечатали какой-нибудь лишний символ. Компилятор VBA знает, что Unload Me вполне завершенный оператор и за ним в строке больше ничего не должно следовать. Как только вы переместите точку ввода в следующую строку, компилятор перепишет весь неправильный оператор красным цветом и выделит посторонний символ. Вы получите также сообщение Compile error. Expected: end of statement (Ошибка компиляции. Ожидался конец оператора).

Компилятор может обнаружить далеко не все ошибки, допущенные вами при печатании программного кола. Когда вы запустите программу, компилятор обнаружит и другие ошибки, но о них мы поговорим немного позже.

Создание второй процедуры

Вторая процедура, которая должна отображать на экране сообщение, чуть сложнее первой. Эта процедура должна вызываться при появлении формы на экране. В окне программного кода, которое должно у вас остаться активным, выполните следующие шаги.

1. **В текстовом поле слева вверху окна программного кода, в котором до сих пор было написано OKButton, щелкните на стрелке в правом конце поля.**
   После этого откроется список объектов, имеющих отношение к форме (рис. 3.16).

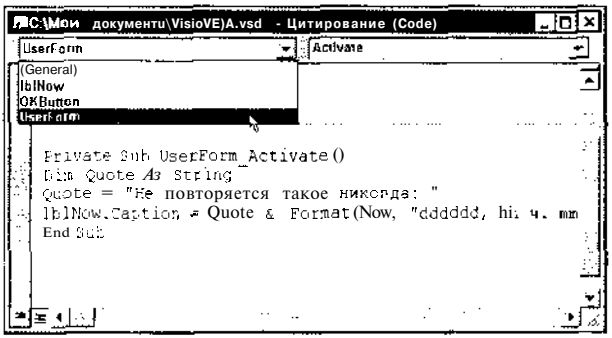


Рис. 3.16. В открывающемся списке объектов перечислены объекты, имеющие отношение к форме, которой принадлежит это окно программного кода

2. **Из раскрывающегося списка выберите UserForm.**
   VBA создаст новую процедуру для события Click (щелчок). Эта процедура будет вызвана, если пользователь вашей программы щелкнет кнопкой мыши в любом месте формы, где нет элементов управления. Нашей с вами программе такая процедура не нужна, поэтому пока оставьте ее.

3. **Теперь щелкните на стрелке в конце текстового поля, находящегося справа, чтобы открыть список процедур, в котором будут перечислены все события, которые VBA распознает для объекта UserForm.**

   Этот список достаточно велик, что дает вам возможность сделать свою программу чуткой практически настолько, насколько захотите (рис. 3.17).

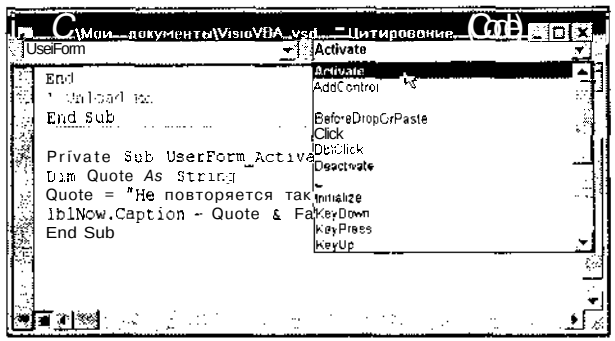


Рис. 3.17. В списке процедур представлены события, которые будут для активного в данный момент объекта

4. **Выберите `Activate` — самый первый пункт в списке. Чтобы увидеть этот пункт, вам придется пролистать список.**

   VBA добросовестно создаст заготовку процедуры `UserForm_Activate`, которая вызывается при загрузке формы в память.

5. **Если вас раздражает пустая процедура `UserForm_Click`, удалите ее, выделив весь ее текст и нажав клавишу <Del>.**

   Этот шаг не обязателен — соответствующий программный код не принесет никакого вреда. Но из-за того, что этот программный код не делает ничего полезного, а только отвлекает, можете смело прервать его одинокую жизнь.

Добавление программного кода в новую процедуру

Поскольку заготовка процедуры и в этом случае, как всегда, создается автоматически, вы сразу можете сосредоточиться на ее наполнении. Между строками, которые любезно предложены вам VBA, напечатайте еще три строки кода. Результат должен выглядеть следующим образом:

```
Private UserForm_Activate()
Dim Quote As String
Quote = "Не повторяется такое никогда: "
lblNow.Caption = Quote & Format(Now, "dddddd, hh ч. mm мин.")
End Sub
```

Первая из напечатанных вами строк

```
Dim Quote As String
```

создает переменную с именем `Quote` и определяет ее как *строковую*, что означает последовательность текстовых символов.

Следующая строка

```
Quote = "Не повторяется такое никогда: "
```

сохраняет **текст** "Не повторяется такое никогда: " в только что созданной переменной `Quote`. Точно так же, как на уроках алгебры в школе, вы здесь используете знак равенства, чтобы приписать определенное значение переменной. Обратите внимание на то, что

перед закрывающими кавычками добавлен пробел, чтобы отделить текст сообщения от следующего за ним текста.

Наконец, строка

```
lblNow.Caption = Quote 5 Format(Now, "dddddd, hh ч. mm мин.")
```

содержит программный код, который непосредственно отображает нужное сообщение в форме. Эта строка начинается с идентификации элемента управления с именем `lblNow` как объекта, с которым идет работа. После имени стоит точка, означающая, что далее идет свойство объекта `lblNow` — в данном случае это свойство `Caption`. Вообще любое свойство подобно переменной, и его можно изменить во время выполнения программы. Поэтому, чтобы изменить установки свойства, в программном коде используется знак равенства. Оставшаяся часть строки определяет сообщение, которое должно появиться на экране в виде надписи.

Первой частью сообщения является переменная `Quote`. Следующий затем знак "плюс" дает указание VBA добавить то, что следует дальше, к тексту, хранящемуся в переменной `Quote`. В скобках функция **Now** говорит VBA, что нужно сбегать и узнать текущие показания часов компьютера, которые постоянно отсчитывают дату и время. Затем функция `Format` берет эту сырую информацию и представляет ее в виде, который сможете прочитать вы. Странный набор букв в кавычках как раз и определяет вил даты и времени на экране, но здесь я не собираюсь вдаваться в детальные объяснения по этому поводу — подробную информацию вы найдете в главе 11.

3-й шаг. Тестирование программы

Так это только шаг 3? Не волнуйтесь, остальные шаги цикла разработки программы не будут слишком длинными.

Запуск программы из редактора Visual Basic

Теперь программа готова к пробному запуску, с помошью которого вы сможете увидеть, работает пи она в соответствии с вашими планами. Чтобы запустить программу из редактора Visual Basic, выполните следующее.

1. **Щелкните либо в окне формы, либо в окне программного кода, чтобы соответствующее окно стало активным.**

   Вспомните, что выяснить, какое из окон активно, можно по полосе заголовка окна — у активного окна цвет полосы заголовка отличается от цвета полос заголовков неактивных окон (у которых по умолчанию они тускло-серые).

2. Затем воспользуйтесь одним из следующих способов.

- ✓ Выберите Run⇨Run Sub/UserForm из меню.
- ✓ Щелкните на кнопке Run (Выполнить) в панели инструментов **Standard** (Стандартная) редактора Visual Basic. (Сразу после установки приложения панель инструментов **Standard** оказывается единственной видимой панелью инструментов в редакторе Visual Basic до тех пор, пока вы сами не измените настройки в соответствии с инструкциями главы 5.)
- ✓ Нажмите <F5>.

После небольшой задержки окно нашей с вами программы появится на фоне вашего VBA-приложения (а не редактора Visual Basic). Если все в порядке, вы увидите нечто, очень похожее на рис. 3.18.

Рис. 3.18. Окноготовойпрограммы-примераво времявыполнения

Не забудьте выполнить указанный выше п. 1, прежде чем запускать VBA-программу! Если не активизировать сначала окно формы или окно ее программного кода, то при попытке запуска программы не случится ничего (или будет запушена какая-нибудь другая программа).

Компиляция

Прежде чем запустить VBA-программу первый раз, ее следует скомпилировать (по поводу определения компиляции снова загляните в главу 1). В процессе компиляции ваша программа готовится к быстротечной жизни, а ее программный кол проверяется строка за строкой на соответствие правилам языка VBA. При первом же отклонении от этих правил компилятор остановится и даст вам шанс исправить ошибку. Даже самая маленькая орфографическая ошибка вызывает экстренную остановку процесса компиляции. Если компилятор обнаруживает ошибку, он выводит на экран небольшое окно с соответствующим сообщением. Среди предлагаемых в этом окне вариантов чаще всего наиболее предпочтительный предлагается кнопкой Debug (Отладка). Щелчок на кнопке Debug переносит вас прямо в окно программного кода, где окажется выделенной строка-обидчик. Компилятор будет помнить результаты своей работы вплоть до выделенной строчки, так что у вас будет возможность исправить ошибку и продолжить компиляцию с того места, где она прекратилась, сделав это любым из упомянутых выше способов для запуска программы.

Если хотите, попытайте счастья и с кнопкой Help (Справка), но не удивляйтесь, если предоставляемая при этом информация окажется слишком общей и поэтому в большинстве случаев бесполезной. Кнопка End (Завершить) прекращает компиляцию и возвращает вас в окно редактора Visual Basic.

Клиническиеиспытания

Может случиться, что, по мнению компилятора, ваша программа совершенно здорова, но это не значит, вообще говоря, что она будет работать именно так, как вы планировали. Отсутствие ошибок при компиляции — это лишь разрешение начать настоящее тестирование.

Хотя компилятор никогда не пропускает ошибок языка VBA, он все же не может читать ваши мысли. Так, компилятор будет вполне доволен строками программного кода, удовлетворяющими все правила языка VBA, даже если в программе эти строки лишены смысла. Представьте себе учителя, который ставит отличную оценку за подобную поэму:

```
Розы красные.
Фиалки фиолетовые.
У меня есть велосипед.
Любишь ли ты рыбу?
```

потому что каждая из строк имеет вполне понятный смысл. Компилятор VBA подобен такому учителю.

Как только компилятор закончит свою работу, программа начнет выполняться. Вы можете теперь испытать все ее возможности и увидеть, работает ли она так, как вы планировали.

Нашу с вами программу тестировать легко. Если в окне будет нужное сообщение, а не Windows Must Die !, и если щелчок на кнопке OK завершает выполнение программы, то примите мои поздравления. Если же нет, то придется вам заняться искоренением ошибок.

4-й шаг. Отладка

По мере усложнения программ вероятность появления ошибок в работе *(ошибок выполнения)* растет экспоненциально. После обнаружения такой ошибки главной задачей становится выявление причины ошибки и устранение проблемы. Одним словом, это отладка.

В небольших программах иногда работает следующий грубый, но достаточно эффективный метод: просто начните внимательно читать весь программный код с самого начала строку за строкой, чтобы попытаться обнаружить, что именно порождает проблему. Поняв это, вы сможете, как правило, без особых усилий устранить ее. (Если что-то не так с вашей программой-примером, просто сравните свой программный код с напечатанным здесь и внесите необходимые исправления.)

К счастью, VBA предлагает несколько средств автоматизации процесса отладки в дополнение к предложенному выше методу ломания головы. Все подробности вы найдете в главе 14.

Вызов формы из VBA-приложения

Знайте, что описанные выше методы — а работают они только в редакторе Visual Basic, — будут единственными способами непосредственного запуска программ, построенных на форме, как наша программа-пример. Чтобы вывести форму на экран из VBA-приложения, нужно с помощью редактора Visual Basic создать процедуру типа Sub (макрос), которая вызывает форму, а затем выполнить эту процедуру (макрос) в приложении. Для этого сделайте следующее.

1. **В редакторе Visual Basic выберите Insert⇨Module, чтобы создать модуль и открыть окно** его **программного** кода.
2. **Напечатайте** следующие **строки программного** кода:

```
Sub ShowQuote()
  UserForm1.Show
End Sub
```

Если ваша форма названа не `UserForm1`, а как-то иначе, замените это имя во второй строке на правильное.

Сохранив изменения, вы сможете начать выполнение макроса ShowQuote способом, предлагаемым в главе 4. Более подробно выполнение программ, основанных на формах, обсуждается в главе 10.

Возвращение в VBA-приложение

Я считаю, что лучший способ доступа к VBA-приложениго при работе в редакторе Visual Basic обеспечивается стандартными приемами Windows для перехода от приложения к приложению. Нажимайте <Alt+Tab> до тех пор, пока в появившемся окне не будет выбрана пиктограмма нужного вам приложения, после чего отпустите клавиши. Любой из следующих способов годится тоже.

- ✓ Выберите View⇨AutoCAD (или имя того приложения, с которым работаете вы).
- ✓ Щелкните на кнопке View в левой части панели инструментов редактора Visual Basic — на этой кнопке будет пиктограмма, представляющая документ приложения.
- ✓ Нажмите <Alt+F11>.
- ✓ Щелкните на кнопке приложения в панели задач Windows.

Вернуться в редактор Visual Basic можно с помощью <Alt+Tab>, панели задач, снова нажав <Alt+F11> или выбрав Сервис⇨Макрос⇨Редактор Visual Basic.

Собираясь пожелать редактору Visual Basic доброй ночи, сделайте это одним из следующих двух способов, каждый из которых закрывает редактор и возвращает вас как раз к тому приложению, откуда этот редактор был вызван.

- ✓ Выберите File⇨Close and Return to *<приложение>*.
- ✓ Нажмите <Alt+Q>.

Глава 4

Выполнение VBA-программ

В этой главе...

- Выполнение программ и макросов из диалогового окна Макрос — надежно, но не слишком интересно
- Запуск макросов с помощью кнопок панели инструментов и пунктов меню
- Назначение для макросов комбинации клавиш
- Автоматический запуск программ при наступлении определенных событий

Убойная сила VBA-программы не значит ничего до тех пор, пока вы не нажмете ее спусковой крючок. Л в нашем случае ЭТОТ спусковой крючок сначала нужно еще найти. Говоря более прозаическим языком, пока вы не найдете способ, с помощью которого сможете запустить свою VBA-программу, программа остается бесполезной.

Во время создания программы в окне редактора Visual Basic вы всегда могли выполнить ее, нажав <F5> или щелкнув на кнопке Run (Выполнить) в панели Standard (Стандартная) редактора Visual Basic. Это вполне удобно при тестировании программы, но совершенно никуда не годится для реальной работы. После того как программа готова, ее нужно запускать из соответствующего приложения.

В этой главе обсуждаются возможности, которые можно использовать для запуска VBA-программ и макросов из VBA-приложений. (Еще раз напомню, что макрос — это та же VBA-программа, которой посчастливилось быть записанной средствами автоматической записи; подробности в главе 2. "Не пишите программу, когда можно записать макрос".) Одна из самых простых возможностей для запуска большинства программ обеспечивается диалоговым окном Макрос, но уж очень это неудобно. VBA-программа становится действительно полезной, когда вы можете запустить ее командой меню, кнопкой в панели инструментов или нажатием определенной комбинации клавиш, а еще лучше, предусмотрев автоматическое выполнение.

Попрактикуйтесь в этом с макросами, которые вы запишете, следуя указаниям главы 2, или с процедурами, которые вы создадите с помощью редактора Visual Basic, используя либо краткие инструкции из главы 6, либо подробные объяснения из частей II и III.

Все определяется именем

Вызов VBA-программы осуществляется по имени.

Извините за то, что приходится начинать с технических подробностей, но другого выхода я не вижу. При вызове VBA-программы вы на самом деле запускаете одну специальную процедуру VBA. Ваша программа может содержать немало самых различных процедур, но только одна из них имеет первую строку кода, говорящую о том, что вы хотите выполнить программ). Эта первая процедура может запускать (вызывать) другие процедуры,

которые в свою очередь могут вызывать следующие и т.д. Для того чтобы выполнить программу, вы должны вызвать только первую процедуру — остальные выполнятся автоматически по мере выполнения программой своей задачи.

Хотите узнать что-нибудь более существенное по поводу процедур? Обратитесь к главе 6, где вы получите полный ответ.

А в чем же здесь проблема? В том, что. запуская VBA-программу, вы должны знать имя процедуры, которая "разбудит" эту программу. В программах, содержащих больше одной процедуры, запускающая процедура обычно называется Main.

Запуск из диалогового окна Макрос

Вполне надежно запускать VBA-программы из диалогового окна Макрос. Если вы не позаботились о том, чтобы приписать программу кнопке в панели инструментов или комбинации клавиш, либо если вы просто забыли, чему вы ее приписали, всегда можно воспользоваться диалоговым окном Макрос.

Вызов диалогового окна Макрос

Чтобы вызвать диалоговое окно Макрос в приложениях Office или в Visio, выполните одно из следующих действий.

- ✓ Выберите команду Сервис⇨Макрос⇨Макросы.
- ✓ Нажмите <Alt+F8>.

Другие VBA-приложения предлагают иные возможности для того, чтобы открыть диалоговое окно Макрос.

Диалоговое окно Макрос должно выглядеть подобно изображенному на рис. 4.1. Как всегда, некоторые детали могут быть другими, в зависимости от используемого вами VBA-приложения.

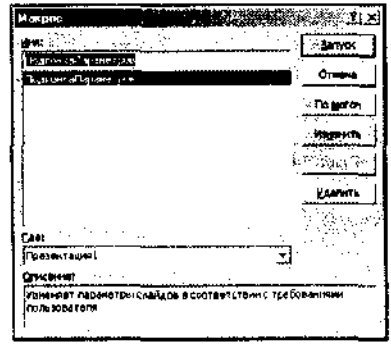


Рис. 4./.ДиалоговоеокноМакрос вприложенииPowerPoint

Больше всего места в диалоговом окне Макрос выделено для списка доступных в данный момент VBA-программ.

Строго говоря, все процедуры, приведенные в списке диалогового окна Макрос, являются процедурами типа Sub (т.е. подпрограммами) без аргументов. (Загляните в главу 2, чтобы убедиться, что это и есть официальное определение макроса.) После запуска такой процедуры *от может* вызывать процедуры других типов, включая процедуры типа Function (т.е. функции) и процедуры Sub с аргументами. Подробности в главе 6.

Выполнение макроса

Запуск макроса из диалогового окна Макрос вряд ли можно назвать большим достижением. Выполняемая при этом последовательность шагов должна быть такой.

1. Выберите макрос из списка ниже поля **Имя.**
2. Щелкните на кнопке **Выполнить.**

Ну как, круто? Как альтернативу можно использовать двойной щелчок на имени программы в списке.

Единственной сложностью при этом может оказаться поиск в списке диалогового окна Макрос имени той программы, которую нужно выполнить. Если вы испытываете такие трудности, обратитесь за помощью к следующему разделу.

Поиск макроса в диалоговом окне Макрос

В диалоговом окне Макрос имена приведенных в списке VBA-программ (макросов) могут иногда сбивать с толку. В зависимости от того, где хранится программа, она может приводиться в списке с идентифицирующим ее префиксом.

Рассмотрим для примера Excel, где вы можете выполнять как программы, хранящиеся в активной рабочей книге (т.е. той. которую вы используете в этот момент), так и программы из других открытых рабочих книг. В диалоговом окне Макрос макросы из активной рабочей книги приводятся под своими собственными именами. А вот макросы из других открытых книг представлены именами, составленными из имени рабочей книги и следующего за ним имени макроса с восклицательным знаком между этими именами в качестве разделителя. Пример показан на рис. 4.2.

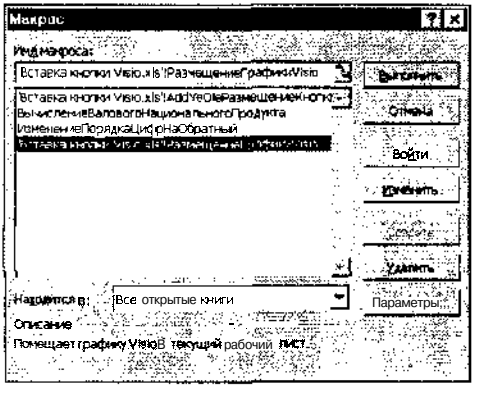


Рис. 4.2. Диалоговое окно Макрос в Excel приводит имена VBA-программ (макросов) из неактивных рабочих книг, любезно указывая перед именем программы имя содержащей ее книги

В диалоговом окне Макрос в Word процедуры приводятся в списке по их именам, *за исключением* того случая, когда две или несколько процедур имеют одно и то же имя. В этом случае имя состоит из имени содержащего процедуру модуля, за которым следуют сначала точка, а затем уже имя процедуры. В качестве доказательства я поместил здесь рис. 4.3.

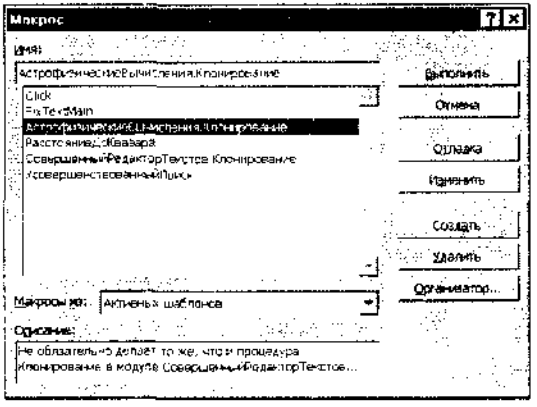


Рис. 4.3. Диалоговое окно Макрос в Word

Если программа, которую вы хотите запустить, отсутствует, вам необходимо выбрать и окне Макрос другой вариант. Для того чтобы это сделать, воспользуйтесь списком Макросы из, в котором выберите вариат Активных шаблонов.

Быстрый запуск программ

Может быть, диалоговое окно Макрос работает и идеально правильно, по оно так неэффективно! Ведь когда требуется что-то выполнить, обычно это нужно *немедленно*. Кто захочет открывать диалоговое окно, искать в длинном списке приведенных мелким шрифтом имен и щелк, щелк, щелк, щелк.

Есть и лучшие способы запуска VBA-программ. Назначьте каждой из СНОПУ очень часто используемых программ кнопку в панели инструментов, пункт в меню или комбинацию клавиш. Тогда один-два щелчка или одна нажатая комбинация клавиш принесут вам желанное вознаграждение.

Как вы, наверное, помните (если, конечно, читали главу 2), многие VBA-приложения позволяют связать макрос с кнопкой в панели инструментов или комбинацией клавиш уже при записи макроса. Это, кстати, довольно веская причина в пользу того, чтобы начинать каждую новую VBA-программу как записываемый макрос, который затем нужно дополнить своим программным кодом. Если вам нужно обновить в памяти информацию о тонкостях этого процесса, обратитесь к главе 2.

После того как макрос записан, у вас тоже должна быть возможность назначить ему кнопку или комбинацию клавиш, а может быть, и пункт меню. Вся остальная часть материала этой главы посвящена обсуждению именно этих возможностей. К сожалению, техника выполнения этой задачи немного варьирует от приложения к приложению. На некоторые из таких вариаций я буду обращать ваше внимание в ходе следующего (довольно долгого) раздела.

Кнопки запуска программ

Какое приятное волнение испытываешь, щелкая на маленькой оригинальной кнопке на экране и получая при этом немедленный ответ своего компьютера! Несомненно, существуют и более приятные волнения, но и нажатие кнопки имеет свою прелесть. А если кнопка, на которой вы щелкаете, вызывает созданную лично вами VBA-программу, к этому волнению добавляется еще и небольшое головокружение от успеха.

В некоторых VBA-приложениях назначение кнопок для вызова VBA-программ происходит очень просто, даже если вы создавали программу, не записывая ее в виде макроса. К таким приложениям относятся приложения из Office. В других VBA-приложениях, как в Visio, вам, вероятно, придется самому печатать программный код VBA, который добавит подходящую кнопку. Ой! А я-то думал, что VBA предлагается в виде визуального средства разработки интерфейса!

Кнопки на панели инструментов или меню

Если вы никак ничего не можете решить по этому поводу, просто помните, что между кнопкой на панели инструментов и элементом меню больших различий нет — все они реагируют на щелчок мыши. Основное отличие состоит в том. что в меню отображается текст, а на кнопке — картинка. Результат использования обоих элементов будет один и то же.

Приложения Office позволяют использовать приемы drag-and-drop для назначения VBA-программе кнопки на панели инструментов или элемента меню.

Простое обращение с кнопками в Office

В Microsoft Word, Excel, PowerPoint, Access, а также Outlook назначать VBA-программе новую кнопку в панели инструментов очень просто — перетащите с помощью мыши. После того как кнопка займет предназначенное ей место, вы можете создать на этой кнопке еще и подходящий рисунок.

Создание кнопки или команды меню в Word

Чтобы в Word создать кнопку для вызова VBA-программы, сделайте следующее.

1. Щелкните правой кнопкой мыши в любой из панелей инструментов.
   После этого на экране возникнет контекстное меню.
2. В самом низу этого меню выберите Настройка, чтобы открыть диалоговое окно Настройка с вкладками, в котором будет изначально открыта вкладка **Панели инструментов.**
   Открыть диалоговое окно Настройка можно без выполнения п. 1, выбрав из меню Сервис⇨Настройка.
3. В диалоговом окне **Настройка щелкните** на ярлыке вкладки **Команды** (рис. 4.4).
4. Прокрутите список **Категории** вниз и найдите пункт **Макросы.** Щелкните на этом пункте, чтобы выбрать его.
   Пои этом справа, в списке Команды, появятся все имеющиеся в вашем распоряжении VBA-программы. (Диалоговое окно Настройка на рис. 4.4 как раз и показывает макросы.)
5. Найдите необходимый макрос в списке (это может быть и VBA-программа, написанная вами с самого начала).
6. Если вы создаете кнопку, найдите в списке ту программу, которой нужно назначить кнопку. Затем перетащите имя этой программы в подходящую панель инструментов. Если вы создаете команду в меню, найдите в списке ту программу, которой нужно назначить кнопку. Затем перетащите имя этой программы прямо в меню.

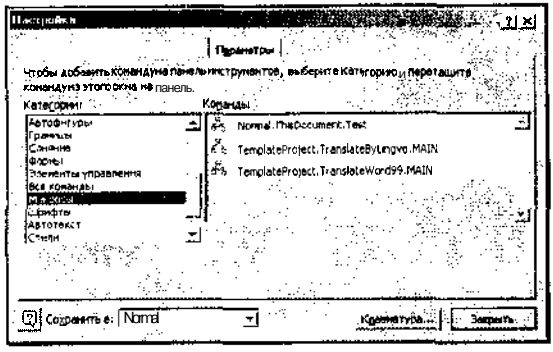


Рис. 4.4. Диалоговое окно Настройка с открытой вкладкой Команды

После того как вы отпустите кнопку мыши, в панели инструментов появится новая кнопка с именем выбранной вами программы на ней.

7. Закройте диалоговое окно **Настройка,** чтобы завершить работу.

Теперь всякий раз после щелчка на этой новой кнопке вызывается для выполнения ваша VBA-программа. Чудеса!

Создание кнопок в FrontPage и Excel

В FrontPage и Excel вам придется выполнять несколько иные действия для создания кнопок запуска VBA-программ. После записи макроса или написания VBA-программы выполните пп. 1–4 предыдущей инструкции. В результате, вместо списка доступных макросов, вы увидите только два элемента в списке Команды диалогового окна Настройка: Настраиваемая команда меню и Настраиваемая кнопка (рис. 4.5). Выполните следующее.

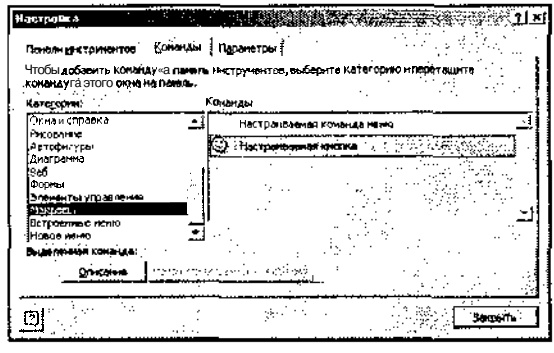


Рис. 4.5. Диалоговое окно Настройка в Excel, готовое к созданию новой кнопки или команды меню

1. Перетащите необходимый элемент, **Настраиваемая команда меню** или **Настраиваемая кнопка,** на нужное место на панели инструментов или меню. Когда вы отпустите кнопку мыши, новый элемент появится на панели инструментов или в меню. Если вы создаете кнопку, на ней будет изображена желтая рожица, если вы создаете команду меню, вы увидите только название макроса.
2. Не закрывая диалоговое окно **Настройка,** щелкните правой кнопкой мыши на новой кнопке меню, чтобы отобразить ее контекстное меню, показанное на рис. 4.6.

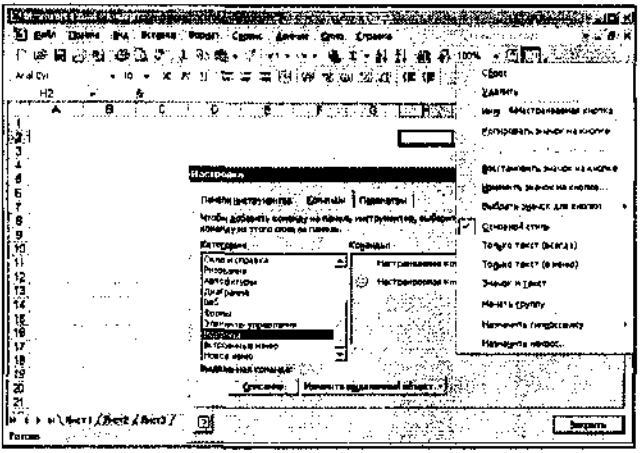


Рис. 4.5. Используйте это меню для назначений VBA-программы кнопке панели задач в Excel или FrontPage

3. Выберите команду **Назначить макрос** из контекстного меню.
   На экране отобразится отдельное диалоговое окно, в котором будут представлены все доступные макросы.
4. Выберите необходимый макрос.
   Выбрав макрос, щелкните на кнопке OK для возвращения к диалоговому окну Настройка, в котором вы сможете выбрать значок для кнопки. Подробности дальше в разделе "Выбор или создание подходящего значка для кнопки".
5. Закройте диалоговое окно **Настройка.**

Создание кнопок на панели задач в Access

Access несколько отличается от других приложений Office, когда дело доходит до назначения VBA-программе кнопки на панели задачи или команды меню. Основное отличие состоит в том, что при создании программ для кнопок или команд вы должны использовать процедуры `Function`, а не процедуры `Sub`, как в других приложениях Office. В главе 6 вы найдете подробное описание этих двух типов процедур.

После написания процедуры `Function`, которая должна выполняться после щелчка на кнопке или выбора команды меню, выполните следующее.

1. **Начните с выбора команды Сервис⇨Настройка, после чего перейдите на вкладку Команды (точно так же, как вы это делаете и в других приложениях Office).**
   Пример диалогового окна **Настройка** на программы Word был приведен на рис. **4.4.**
2. **В списке Категории, расположенном в левой части окна, оставьте выбранным вариант Файл.**
3. **Перетащите первый элемент из списка команд в необходимое место на панели инструментов или в меню (точно так же, как вы это делаете и в других приложениях Office).**
4. **Перетащив, щелкните правой кнопкой мыши на полученной кнопке или команде, чтобы отобразить контекстное меню.**
   Подобное контекстное меню было показано раньше на рис. 4.5.
5. **Выберите** команду **Свойства, чтобы отобразить диалоговое окно свойств.**
   Внешний вид этого диалогового окна уникален для Access.

6. **В текстовом поле действия укажите имя VBA-программы (процедуры Function).**
   Вы должны указать перед именем программы знак равенства, а после него — скобки, как показано ниже.
   ```
   =Имя_функции ()
   ```
7. **Закройте диалоговое окно свойств.**
   Вы можете настроить внешний вид кнопки, применив приемы, описанные дальше в разделе "Выбор или создание подходящего значка для кнопки".

Настройка панелей задач в других ***VBA-приложениях***

Другие VBA-приложения значительно отличаются по способу создания кнопок на панели инструментов. Например, в Visio вам придется написать VBA-код, выполняющий всю необходимые действия; в то же время в CorelDraw вам необходимо выполнить действия, очень похожие на те, которые вы выполняете в Word, PowerPoint и FrontPage.

1. **Начните с выбора команды Tools⇨Options (Сервис⇨Параметры).**
2. **Из списка, расположенного в левой части окна, выберите Commands** (Команды).
3. **Выберите Macros (Макрос) из раскрывающегося списка в средней части диалогового окна.**
   Теперь вы можете перетащить на панель задач любой из доступных макросов.

Выбор или создание подходящего значка для кнопки

Ясно, что суперширокие кнопки с надписями типа

```
VBAДляЧайников.ГрафическиеПрограммы.СозданиеИзображения
```

не только выглядят отвратительно, но попросту зря занимают драгоценное место в панели инструментов. К счастью, вы легко можете заменить эту многословную надпись на компактное графическое изображение. Для этого сделайте следующее.

1. **Откройте диалоговое окно Настройка и** перейдите **на вкладку Команды (см. инструкции из раздела "Создание кнопок на панели задач в Access", пп. 1-3).**
2. **Щелкните правой кнопкой мыши на своей новой кнопке в панели инструментов.**
   Появится весьма многословное контекстное меню.
2. **Примерно посередине этого меню есть пункт Выбрать значок для кнопки, выберите его, чтобы открылось графическое меню изображений для кнопок.**
   Результат выполнения этой процедуры показан на рис. 4.7.
4. **Если вас устроит какое-то из предлагаемых изображений, сразу переходите к п. 8, а иначе продолжайте по порядку.**
5. Выберите в контекстном меню пункт Изменить значок на кнопке.
   Появляющееся при этом окно редактора кнопок— небольшой студии, в которой можно по точкам менять изображение для кнопки, показано на рис. 4.8.
6. **Выберите цвет из палитры цветов, а затем щелкните в нужной вам ячейке увеличенного изображения, чтобы закрасить ее выбранным цветом.**
   Чтобы удалить закрашенную часть изображения и открыть в соответствующем месте изображения поверхность кнопки, перед закрашиванием щелкните в квадратике Удалить. Щелчок на кнопках Перемещение позволяет сдвинуть изображение в сторону, где еще нет закрашенных ячеек.
   Теперь на кнопке есть изображение и можно удалить с нее текст.
7. **Снова откройте контекстное меню (см. п. 2).**
8. **Установите флажок Основной стиль.**

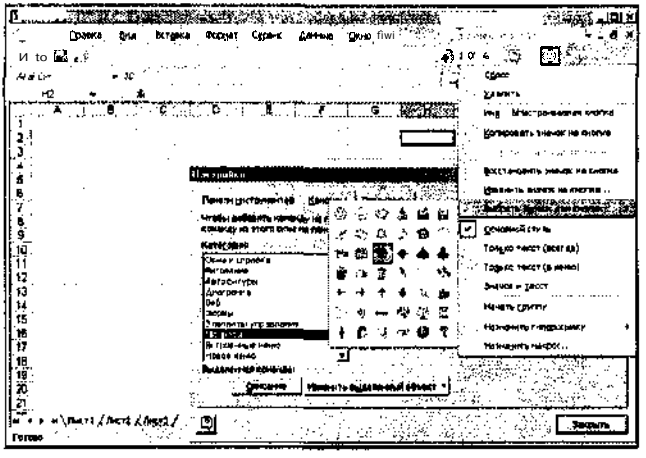


Рис. 4.7. При открытом окне Настройка щелчок правой кнопкой в панели инструментов открывается показанное здесь контекстное меню

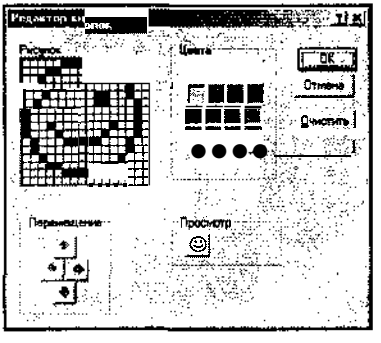


Рис. 4.8. Используйте предлагаемый Office редактор кнопок для создания своих собственных изображений на кнопках

Как только вы закончите вышеописанную процедуру, в панели инструментов уверенно обоснуется новая симпатичная кнопка, готовая к действию после каждого вашего щелчка на ней.

Удаление кнопок

В Office можно удалить ненужную кнопку, перетащив ее за пределы панелей инструментов. Хитрость здесь заключается в том, что при перетаскивании нужно нажать и удерживать клавишу <Alt>. Когда рядом с указателем мыши возникнет значок X, отпустите кнопку мыши, и удаление произойдет.

Настройка кнопок с помощью VBA-кода

Чувствую я, что ваше VBA-приложение как раз и не позволяет просто перетаскивать нужное при настройке кнопок. Необходимость использовать VBA при создании кнопок в самом начале знакомства с VBA, наверное, выглядит устрашающе. Но на самом деле такой способ создания кнопок не слишком сложен. Главное — выяснить, какими объектами в вашем VBA-приложении представляются кнопки и панели инструментов и какие свойства и методы нужно использовать для изменений.

И они еще называют VBA стандартом!

В Office объекты кнопок называются `CommandBarButtons`, а принадлежат они коллекции `CommandBarControls`. Объект `CommandBar` (панель инструментов, содержащая коллекцию `CommandBarControls`) размещается в коллекции `CommandBars`, которое может принадлежать объекту `Application` или объекту `Document`.

А в Visio, например, отдельная кнопка представляется объектом `ToolbarItem`. Однако, чтобы воспользоваться конкретным экземпляром объекта `Toolbarltem`, необходимо получить доступ к той коллекции `ToolbarItems`, которой этот экземпляр объекта принадлежит. Сама же коллекция `Toolbarltems` принадлежит коллекции `Toolbars`, в свою очередь принадлежащей коллекции `ToolbarSet`, а уж последняя принадлежит конкретному экземпляру объекта `UIObject` (объект пользовательского интерфейса). Немного позже в этой же главе с помощью несложного программного кода я покажу, как все это реализуется практически.

Если все эти сложности вас пугают, успокойтесь: вам не придется создавать программный код для своих кнопок с чистого листа. Поищите в справочной системе своего VBA-приложения разделы, посвященные настройке панелей инструментов, и вы наверняка обнаружите там пример, который после несложных модификаций можно использовать в вашем конкретном случае.

Комбинации клавиш в Word, Excel и Access

Если вы хоть немного умеете печатать, то часто используемые вами команды просто умоляют вас назначить им комбинации клавиш. В большинстве приложений, даже в тех, которые ориентированы исключительно на работу с изображениями, нажать клавиши часто быстрее, чем перемещать туда-сюда мышь и щелкать ее кнопками. Единственный недостаток комбинаций клавиш — необходимость запоминать их.

В любом случае некоторые VBA-приложения — в том числе Word, Excel и Access — дают прямой доступ к клавиатуре, позволяя назначить комбинации клавиш VBA-программам. Если вам предлагают такой способ управления, используйте его, не сомневаясь.

Назначение комбинаций клавиш программам в Word

Если уж зашла речь о настройке клавиатуры, то изо всех VBA-приложений самым гибким и простым в этом отношении является Word. Например, чтобы назначить в Word программе с именем `DoubleCurrentFontSize` комбинацию клавиш <Alt+Shift+.> (т.е. <Alt+Shift+клавиша с точкой>), действуйте следующим образом.

1. Откройте диалоговое окно **Настройка,** для чего либо выберите **Сервис⇨Настройка** из меню, либо щелчком правой кнопкой мыши в любой из панелей инструментов откройте контекстное меню и затем выберите пункт **Настройка** в **нем.**
2. Щелкните на кнопке **Клавиатура** в нижней части вкладки **Параметры.** Появится диалоговое окно Настройка клавиатуры (рис. 4.9).
3. В списке **Категории** выберите Макросы.
4. Найдите и выберите свою программу в появившемся справа списке **Макросы.** Если программе уже была назначена комбинация клавиш, она будет в поле Текущие сочетания клавиш.
5. Нажмите комбинацию клавиш, которую *вы* хотите назначить своей программе. Если предложенная вами комбинация уже используется, диалоговое окно сообщит, какую из команд она вызывает, и вы сможете либо оставить в силе имеющееся назначение, либо заменить его новым.

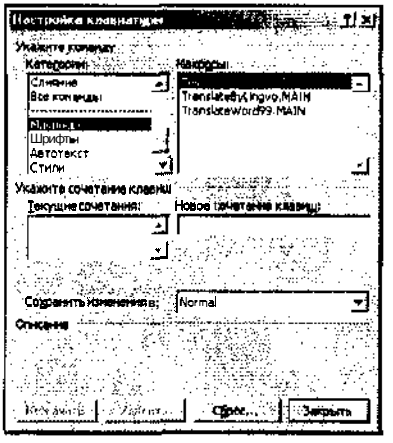


Рис. 4.9. С помощью такого диалогового окна вы можете назначить свои VBA-программам комбинации клавиш в Word

6. Щелкните сначала на кнопке **Назначить,** а затем на кнопке Закрыть, чтобы выйти из диалогового окна.

С этого момента назначенная вами комбинация клавиш будет готова к действию.

Разместите свои программы в контекстных меню

Приложив совсем немного усилий, вы сможете поселить свою скромную VBA-программу в фешенебельных апартаментах любого контекстного (т.е. вызываемого щелчком правой кнопки мыши) меню.

В Office контекстные меню настраиваются в диалоговом окне Настройка. Сначала нужно сделать видимой панель инструментов Контекстные меню. Откройте диалоговое окно Настройка, выбрав Сервис⇨Настройка, и в нем перейдите на вкладку Панели инструментов (если она еще не открыта), щелкнув на соответствующем ярлыке в верхней части диалогового окна. Пролистайте вниз список панелей инструментов, найдите в нем пункт Контекстные меню и щелкните в маленьком квадратике около него.

Панель инструментов Контекстные меню станет видимой. С помощью этой панели инструментов вы сможете получить доступ к любому контекстному меню, чтобы настроить его. Открыв таким образом нужное вам контекстное меню, используйте обсуждавшиеся в начале этого раздела приемы перетаскивания макросов из вкладки Команды диалогового окна Настройка.

Например, Visio позволяет поместить макрос в контекстное меню любой из форм. Выделив форму, выберите Window⇨Show ShapeSheet. В окне ShapeSheet щелкните в ячейке Action. (Если нужно, вставьте перед этим раздел Actions, выбрав Insert⇨Section и отметив Actions.)

Затем выберите Edit⇨Action. Вы получите возможность впечатать текст и определить другие свойства пункта меню и назначить ему программу для выполнения. По окончании редактирования это контекстное меню, вызываемое щелчком правой кнопки мыши на данной форме, уже будет включать и пункт для вашей VBA-программы.

Комбинациинаклавишв Excel

Excel тоже позволяет назначить VBA-программам комбинации клавиш, но ограничивает допустимые комбинации сочетаниями клавиши <Ctrl> с печатаемыми символами (буквами, числами и знаками пунктуации). Чтобы назначить программе комбинацию клавиш, выполните следующее.

1. **Откройте диалоговое окно Макросы (почему бы для этого не воспользоваться комбинацией клавиш <Alt+F8>?)**
2. **Найдите и выделите свою программу в списке.**
3. **Щелкните на кнопке Параметры.**
4. **В появившемся диалоговом окне (рис. 4.10) нажмите клавишу, которую хотите скомбинировать с клавишей <Ctrl>.**
   Excel при этом различает буквы верхнего и нижнего регистров. (Другими словами, клавиша с буквой, нажатая вместе с клавишей <Shift>, отличается от клавиши с буквой, нажатой без клавиши <Shift>.)
5. **Щелкните на кнопке ОК, чтобы вернуться в диалоговое окно Макросы.**

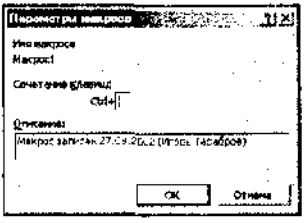


Рис4.10. Назначение VBA-программе комбинациинаклавишв Excel

Комбинациинаклавишв Access

В приложениях баз данных Access назначать комбинации клавиш совсем несложно. Приложения Access построены на использовании форм и элементов управления в этих формах. Все, что вам потребуется сделать, это создать процедуры обработки событий для этих форм и элементов управления— процедуры, выполняемые при нажатии клавиши пользователем. Процедуры обработки событий в мельчайших подробностях описываются в главе 10.

Другое дело, если вы работаете с базой данных Access напрямую, используя команды меню и кнопки панелей инструментов. В этом случае для создания комбинаций клавиш, вызывающих VBA-программы, вам придется назначать эти комбинации клавиш специальным макросам Access. (Напомню, что макросы Access не являются процедурами VBA.) В Access нет средства записи макросов, поэтому вам придется строить макросы вручную в соответствии со следующими инструкциями.

1. **В окне базы данных выберите кнопку Макросы, а затем щелкните на кнопке Создать.**
   Появится диалоговое окно Макрос (рис. 4.11).
2. В столбце **Макрокоманда** выберите **ЗапускПрограммы, как** показано на рис. **4.11.**
3. Выберите **Вид⇨Имена макросов,** чтобы в окне появился новый столбец **Имя макроса.**
4. Напечатайте **в столбце Имя макроса комбинацию клавиш, которую вы хотите** назначить своей **VBA-программе.**

Используйте символ ^ для обозначения клавишм <Ctrl>. а символ + — для обозначения <Shift>. Буквы и числа можно печатать как обычно, а вот функциональные клавиши нужно заключить в фигурные скобки. Например, для комбинации клавиш <Ctrl+k> нужно напечатать ^k,а для <Shift+F8> •- -(F8).

5. **Щелкните в текстовом поле Имя функции и напечатайте имя той процедуры VBA, которую должен вызывать макрос.**
   Это должна быть процедура типа `Function`.
6. **Закройте окно Макрос, щелкнув на кнопке в правом верхнем углу окна. Когда Access попросит дать макросу имя, напечатайте `AutoKeys`.**

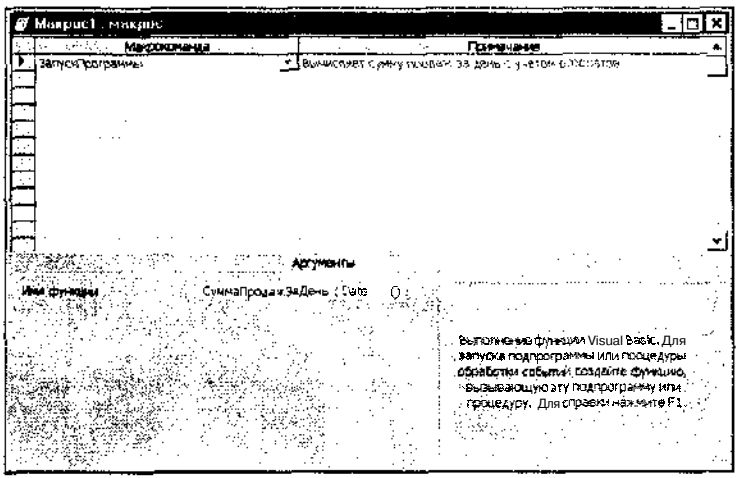


Рис. 4.1 /. Такое окно используется для создания макросов в Access

Макросы для мыши Visio

Visio предлагает совершенно другой способ вызова VBA-программ — это двойной щелчок на форме, которую вы связали со своим программным кодом. Вот инструкции для назначения форме ответа на двойной щелчок кнопкой мыши.

1. **Щелкните на форме, чтобы выделить ее.**
2. **Выберите Format⇨ Behavior.**
3. **Перейдите на вкладку Double-Click.**
4. **Выберите VBA-процедуру своей мечты из списка Run Macro.**
5. **Щелкните на кнопке OK.**

Теперь после двойного щелчка кнопкой мыши на этой форме всегда будет выполняться ваша VBA-программа.

Автоматический запуск VBA-программ

Предположим, что у вас есть VBA-программа, которую вы любите настолько, что хотели бы видеть ее в действии самой первой каждый раз, когда запускаете свое приложение. А если вы любите свою программу так сильно, то и такую, казалось бы, ничтожную задержку, как щелчок на кнопке, наверное, нелегко пережить.

К счастью для вас, многие VBA-приложения предусматривают возможность автоматического выполнения программного кода VBA при запуске и закрытии приложения, при открытии и закрытии документа, а также при наступлении некоторых других событий. В табл. 4.1-4.3 представлены приемы, которые можно использовать в самых популярных VBA-приложениях для автоматического запуска VBA-программ.

Таблица 4.1. Автоматическое выполнение VBA-программ при запуске приложения с помощью ярлыка

Приложение	Команда, добавляемая в поле Файл на вкладке Ярлык	Комментарий
Microsoft Word	/mVBA	Вместо VBA напечатайте имя своей VBA-программы
Microsoft Access	/xмакрос	Выполняет все макросы из указанного вами списка. Чтобы выполнить процедуру VBA, макрос Access должен включать макрокоманду ЗапускПрограммы

Таблица 4.2. Автоматическое выполнение VBA-программ в Word с помощью назначения программам специальных имен

Имя VBA-программы	Условие автоматического выполнения программы
AutoExec	При запуске Word
AutoNew	При создании каждого нового документа
AutoOpen	При открытии любого существующего документа
AutoClose	При закрытии документа
AutoExit	При выходе из Word

Замечание. Макрос AutoExec должен храниться в шаблоне Обычный (normal.dot).

Таблица 4.3. Выполнение VBA-программ с помощью процедур обработки событий (создание процедур обработки событий обсуждается в главе 14)

Приложение	Условие автоматического вызова процедуры	Используйте объект	Создайте процедуру обработки события
Visio	При создании нового документа	Document	DocumentOpened
Visio	При сохранении документа под другим именем	Document	DocumentSavedAs
MicrosoftWord	При открытии документа	Document	Open
MicrosoftWord	При создании нового документа	Document	New
MicrosoftWord	При закрытии документа	Document	Close

Окончание табл. 4.3

Приложение	Условие автоматического вызова процедуры	Используйте объект	Создайте процедуру обработки события
Microsoft Excel	При открытии любой рабочей книги	Application	WorkbookOpen
Microsoft Excel	При открытии конкретной рабочей книги	Workbook	Open
Microsoft Excel	При создании новой рабочей книги	Workbook	NewSheet

Замечание. Совсем немногие из длинного списка событий Visio, Word и Excel относятся ко всему документу.

Работа с ярлыками Windows

Некоторые приложения позволяют выполнить программный код VBA с помощью добавления команд к имени приложения при его запуске (см. табл. 4.1). Если вы не хотите утруждать себя набиранием этих команд в командной строке DOS, модифицируйте или создайте подходящий ярлык Windows, который будет вызывать это приложение.

Пиктограммы в меню Пуск Windows представляют ярлыки. Чтобы модифицировать их, щелкните правой кнопкой мыши на кнопке Пуск, из появившегося небольшого меню выберите пункт Открыть, а затем откройте папку Программы и подходящие вложенные в нее папки, чтобы увидеть ярлык вашего приложения. Теперь щелкните правой кнопкой мыши на ярлыке и выберите из появившегося меню пункт Свойства, чтобы открыть диалоговое окно Свойства.

В этом диалоговом окне перейдите на вкладку Ярлыки. В текстовом поле Файл после имени самого приложения напечатайте нужную команду. Например, чтобы при запуске Word выполнялась процедура VBA, текст в поле Файл должен выглядеть примерно так:

```
C:\Microsoft Office\WINWORD.EXE /mAutomaticNovel
```

Чтобы создать *новый* ярлык, щелкните правой кнопкой мыши в любом месте рабочего стола, в окне Мой компьютер или окне Проводник. В появившемся контекстном меню выберите Создать⇨Ярлык и позвольте Windows провести вас через оставшиеся шаги. После этого появившийся ярлык вы сможете изменить в соответствии с инструкциями, приведенными в этой врезке выше.

Глава 5

Редактор Visual Basic к вашим услугам

В этой главе...

- Поиск команд в системе меню редактора Visual Basic
- Отображение, перемещение и настройка панелей инструментов
- Понимание и использование закрепления панелей инструментов и окон
- Сражение с толпами окон редактора Visual Basic
- Знакомство с комбинациями клавиш, используемыми редактором Visual Basic
- Панорамный обзор VBA-проекта с помощью проводника проектов
- Исследование объектов проекта с помощью обозревателя объектов
- Печатание программного кода — и получение его готовым — в окне программного кода

Помните сцену из мультика, где Мики Маус играет ученика чародея? Волшебная метла, выполняющая всю домашнюю работу, поначалу кажется ему неплохой идеей, но довольно скоро выходит из-под контроля команда ведер, и бедный Мики едва не тонет.

С первых шагов в редакторе Visual Basic тоже может покачаться, что вы тонете. В нем можно очень быстро открыть так много непривычных с виду окон, что впору залиться слезами от отчаяния. Но не паникуйте — через эти окна на ваш ковер не попадет ни капли воды, напротив, каждое из них может сделать часть важной работы за вас. Немного опыта и несложные навыки, которым учит эта глава, позволят вам полностью подчинить себе и редактор Visual Basic, и все ею окна.

Наверное, вы догадываетесь (или опасаетесь), что каждое окно в редакторе Visual Basic должно играть исключительно важную (я бы рискнул сказать, жизненно важную) роль в процессе программирования в VBA. В этой главе описаны все типы окон редактора Visual Basic и даже показано, как практически использовать некоторые из них.

В частности, я собираюсь достаточно подробно обсудить окна Project Explorer (Окно проводника проектов), Object Browser (Окно обозревателя объектов) и Code (Окно программного кода) — и не только внешний вид и управление ими, но и то, как их использовать в процессе программирования. Детальным обсуждением других окон, среди которых UserForm (Окно формы), Properties (Окно свойств), а также четыре окна для отладки программы, мы займемся в главах 9 и 10, но в этой главе вы найдете вводную информацию и об этих окнах тоже.

Пользовательский интерфейс редактора Visual Basic

Редактор Visual Basic является стандартным блюдом Microsoft — меню, панели инструментов и комбинации клавиш выглядят и работают очень похоже на Microsoft Office. Вы будете чувствовать себя как дома, если используете VBA с приложениями из Office.

С другой стороны, вы почувствуете явный контраст между интерфейсом не-Microsoft приложения и интерфейсом редактора Visual Basic. Причина в том, что сторонние производители программных продуктов добавили VBA **R** СВОИ приложения но лицензии *Microsoft,* но сами-то приложения начинали развиваться и оформились задолго до того, как появилась текущая версия Office.

Легкий завтрак с меню

Я уверен, что вы уже знаете, как использовать меню. Тем не менее, наверное, вам будет интересно узнать, какие команды есть в меню редактора Visual Basic и где именно они находятся.

По большей части команды меню редактора Visual Basic организованы логично. Например, если вы хотите открыть какое-то окно, без сомнения, нужная команда будет не где-нибудь, а в меню View (Вид). Однако есть и менее очевидные моменты. Следующая небольшая таблица поможет вам найти команды, затаившиеся в меню.

Задача	Команда меню	Комментарий
Управление установками для всего проекта	Tools⇨Project Properties	Это странное место для команды, относящейся ко всему проекту, - я ожидал бы найти ее в меню File
Настройка Visual Basic	View⇨Toolbars⇨ Настройка⇨Параметры	
Включение режима проектирования	Run⇨Design Mode	Команда Design Mode, похоже, относится к разметке форм, так почему же она тогда в меню Run? Потому что она останавливает выполнение любой программы. Эта команда не слишком полезна, поскольку можно проектировать формы и без нее, и совсем не обязательно выходить из режима проектирования, чтобы выполнить программу

Прогулка по панелям инструментов

Если вы знакомы с Word, Excel или PowerPoint, вероятно, и работа с панелями инструментов в редакторе Visual Basic покажется вам вполне удобной. Если же нет, по этому поводу достаточно будет нескольких кратких комментариев.

Редактор Visual Basic предлагает четыре готовые панели инструментов.

- ✓ **Standard** (Стандартная). Это единственная панель инструментов, которую вы видите при первом запуске редактора Visual Basic. Ее кнопки выполняют самые разнообразные функции: сохранение результатов вашей работы, добавление новых форм и модулей, редактирование и выполнение программ.
- ✓ **Edit** (Правка). Кнопки этой панели инструментов пригодятся при редактировании программного кода. Они дублируют команды меню Edit.
- ✓ **Debug** (Отладка). Здесь размещаются кнопки для команд, которые понадобятся вам при вылавливании ошибок в программах.
- ✓ **UserForm** (Пользовательская форма). Эта панель инструментов используется при проектировании форм. Большинство ее кнопок дублируют команды из меню **Format** для выравнивания, упорядочения и группировки элементов управления в форме.

Эти панели инструментов можно использовать в любой комбинации. Всего пара щелчков отображает панель инструментов на экране (если она еще невидима) или прячет ее (если она присутствует на экране). Вот описание соответствующей процедуры.

1. **Сначала шелкните правой кнопкой мыши в любой панели управления.**
   Появится контекстное меню (рис. 5.1) со списком всех имеющихся панелей инструментов. Если панель инструментов видима, ее имя будет отмечено галочкой.

Рис. 5.1. С помощью этого контекстного меню можно отображать на экране и прятать панели инструментов

2. **В этом меню выберите панель инструментов, которую хотите отобразить или спрятать.**
   Как только вы отпустите кнопку мыши, на экране произойдут соответствующие изменения.

Панели инструментов редактора Visual Basic могут существовать в одном из трех состояний, как показано в следующей таблице.

Состояние панели инструментов	Вид панели инструментов на экране
Скрытая	Невидима на экране
Закрепленная	Прикреплена к одной из четырех сторон рабочей области главного окна
Перемещаемая	Представлена в виде своего отдельного окна с полосой заголовка и закрывающей окно кнопкой. Перемещаемые панели инструментов можно разместить и вне окна редактора Visual Basic

На рис. 5.2 показано окно редактора Visual Basic с двумя закрепленными и двумя перемещаемыми панелями инструментов. Скрытыми, закрепленными или перемещаемыми могут быть большинство основных окон редактора Visual Basic.

Чтобы сделать закрепленную панель инструментов перемещаемой, просто перетащите ее от края рабочей области окна к его середине. Самое удобное место для захвата закрепленной панели инструментов — место с краю, где видны две вертикальные (или горизонтальные) полоски. Эта область панели инструментов называется *маркером перемещения*. Сделать панель инструментов перемещаемой можно и иначе — дважды щелкнув в области *маркера перемещения*.

Обратное тоже верно: чтобы превратить перемещаемую панель инструментов в закрепленную, перетащите ее полосу заголовка к любой из сторон рабочей области окна редактора Visual Basic. Как только указатель мыши окажется достаточно близко, гравитационное поле окна притянет панель инструментов, как магнит. Этот эффект завораживает, но двойным щелчком панель инструментов закрепляется быстрее (она возвращается на то место, где была закреплена в последний раз).

Настройка панелей инструментов и меню

Предположим, что вас почему-то не устраивают панели инструментов, предлагаемые редактором Visual Basic. Ну, так создайте себе новые! Можно добавлять и убирать кнопки и пункты меню в существующих панелях инструментов и меню, а если и этого для вас недостаточно, вы имеете возможность создать свою собственную панель инструментов просто "с нуля" (это делается точно так же, как и в приложениях Office).

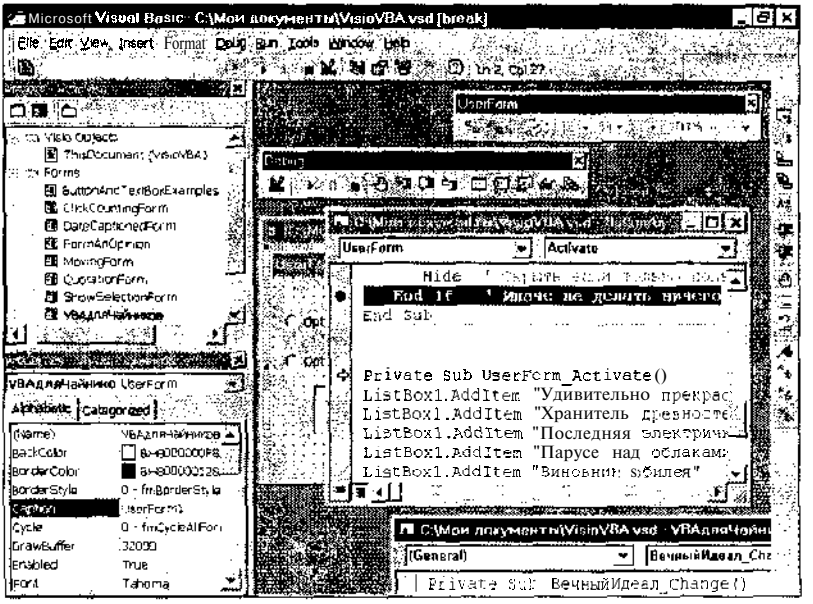


Рис.5.2 По умолчанию панели инструментов редактора Visual Basic закреплены под строкой меню, но их можно закрепить и вдоль других сторон главного окна или сделать свободно перемещаемыми

Перемещение и удаление кнопок в панели инструментов

Если вам нужно только переместить или удалить уже имеющуюся кнопку, то все, что вам потребуется, это не забыть нажать и удерживать клавишу <Alt>, пока вы будете перетаскивать кнопку на новое место. При нажатой клавише <Alt> вы можете сделать следующее.

- ✓ Перетащить кнопку на новое место в той же или другой панели инструментов. Если вам нужно поместить в новое место *копию* кнопки, во время перетаскивания удерживайте нажатыми клавиши <Alt> и <Ctrl>.
- ✓ Добавить линию разделителя между двумя кнопками, перетащив кнопку справа еще чуть-чуть правее (если перетащить еще правее, кнопка перескочит за соседнюю). Чтобы убрать линию разделителя, перетащите правую кнопку немного влево.
- ✓ Удалить кнопку, перетащив ее на любое место, где нет панелей инструментов. Когда рядом с указателем появится значок X, отпустите кнопку мыши, и кнопка из панели инструментов уйдет в историю.

Добавление новой кнопки в панели инструментов

Для более изысканных модификаций потребуется открыть диалоговое окно Настройка. Оно используется для добавления новых кнопок в панели инструментов и для любых изменений в меню.

Вот последовательность действий, необходимых для добавления новой кнопки в панели инструментов.

1. Щелкните правой кнопкой мыши в любой панели **инструментов и в** самом низу появившегося контекстного меню выберите **Настройка.**
   Появится диалоговое окно Настройка (*рис.* 5.3).

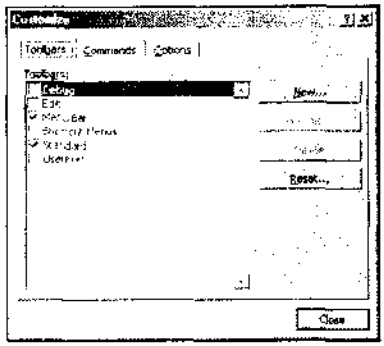


Рис. 5.3. С помощью диалогового окна настройки можно внести радикальные изменения в пользовательский интерфейс редактора Visual Basic

2. Если панели инструментов, которую вы собираетесь изменить, нет на экране, **перейдите** в диалоговом окне **Настройка** на вкладку **Панели инструментов.**
   Установите флажок рядом с нужной вам панелью инструментов. Здесь же можно добавить и новую панель инструментов, для чего нужно щелкнуть на кнопке Создать и набрать имя.
3. Когда панель **инструментов, с которой** вы собирались работать, появится на экране, перейдите на вкладку **Команды.**
   Теперь можно перетаскивать объекты из списка в панели инструментов или меню.
4. После того как вид панели инструментов будет удовлетворять вас, закройте диалоговое окно Настройка.

Обратите внимание, при открытом диалоговом окне Настройка перетаскивать и удалять кнопки из панелей инструментов можно и без помощи клавиши <Alt>. Однако, чтобы скопировать, а не переместить выбранный объект, придется нажать клавишу <Ctrl>.

Обезьянничание с меню

Чтобы изменить меню редактора Visual Basic, необходимо открыть диалоговое окно Настройка (см. п. 1 в предыдущем разделе). После этого вы сможете щелчком на меню открыть его, а затем перетащить интересующие вас пункты меню в другие позиции или другие меню. Перетаскивая пункт меню в рабочую область окна, можно удалить этот пункт меню. Чтобы добавить или удалить разделительную линию между двумя пунктами меню, перетащите нижний из этих пунктов немного ниже или немного выше.

Чтобы добавить в меню новые пункты, перетащите их с вкладки Команды диалогового окна Настройка точно так же, как вы это делали при создании новых кнопок в панели инструментов (рис. 5.3 и 5.4).

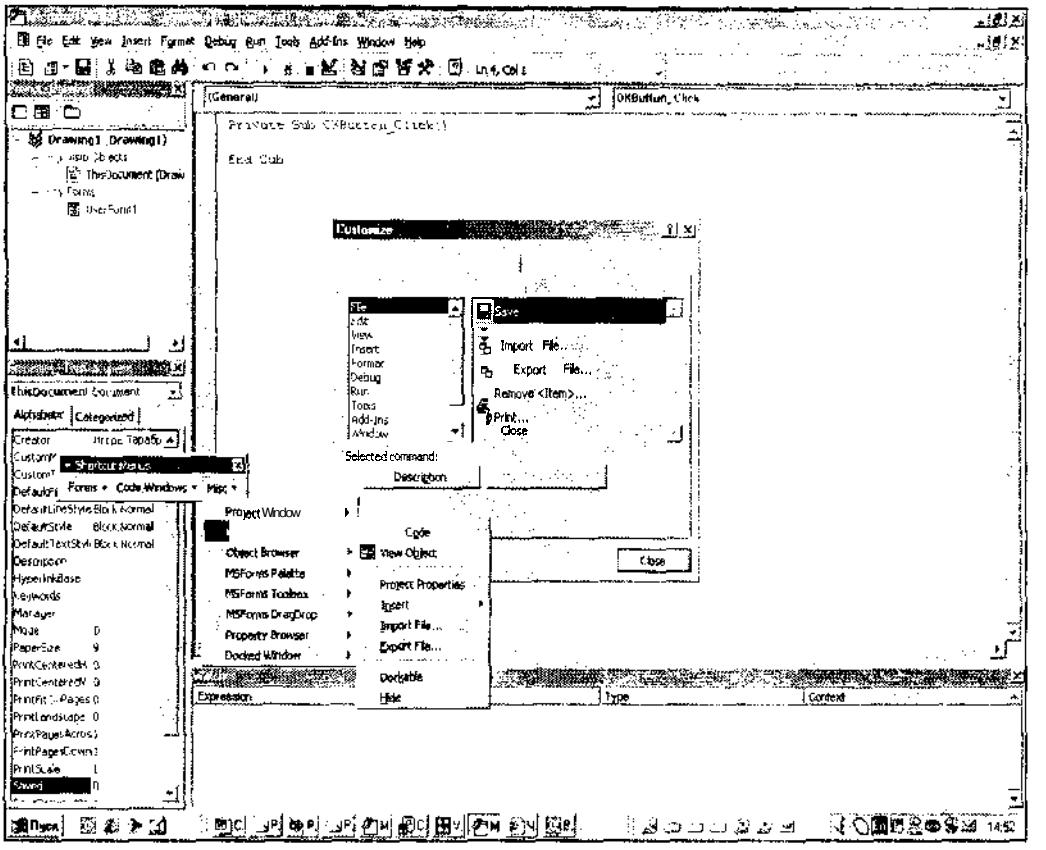


Рис. 5.4. Настройка одного из контекстных меню редактора Visual Basic

Наверное, считая возможность изменения системы главного меню недостаточной, редактор Visual Basic позволяет также изменять контекстные (т.е. вызываемые щелчком правой кнопки мыши) меню. Для этого нужно открыть диалоговое окно Настройка, перейти в нем на вкладку Панели инструментов и отметить флажок Контекстные меню.

В результате этих действий на экране появится небольшая панель инструментов, предназначенная только для настройки (см. рис. 5.4). Найдите в этой панели инструментов нужное вам контекстное меню и щелкните на нем. После этого вы сможете перетаскивать к это контекстное меню объекты с вкладки Команды.

Комбинации клавиш

В табл. 5.1 приведены все комбинации клавиш, задействованные в редакторе Visual Basic. Кроме них, вы можете использовать клавишные команды Windows для управления курсором и редактирования текста. Не забывайте и о том, что <Shift+F10> вызывает контекстное меню для окна или другого объекта, активного в данный момент, — как будто вы щелкаете на этом объекте правой кнопкой мыши.

Таблица 5.1. Комбинации клавиш, предлагаемые редактором Visual Basic

Отображение окон

Для того чтобы...	нажмите...
Открыть окно программного кода для данной формы или элемента управления	<F7>
Отобразить форму, соответствующую активному окну программного кода	<Shift+F7>
Перейти в следующее окно программного кода или в окно формы	<Ctrl+Tab>
Открыть окно обозревателя объектов	<F2>
Открыть окно свойств	<F4>
Открыть окно немедленного выполнения команд	<Ctrl+G>
Открыть окно стека вызовов во время выполнения программы	<Ctrl+L>

Работа с программным кодом

Для того чтобы...	нажмите...
Перейти к определению объекта в точке ввода	<Shift+F2>
Открыть диалоговое окно поиска	<Ctrl+F>
Найти далее (найти, где следующий раз появляется текст, заданный в окне поиска)	<F3>
Найти предыдущее	<Shift+F3>
Заменить	<Ctrl+H>
Перейти к предыдущей редактировавшейся строке	<Ctrl+Shift+F2>
Отменить действие	<Ctrl+Z>
Открыть список свойств/методов	<Ctrl+J>
Открыть список констант	<Ctrl+Shift+J>
Получить краткую справку о переменной или объекте в точке ввода	<Ctrl+I>
Отобразить информацию о параметрах функции в точке ввода	<Ctrl+Shift+I>
Автоматически дополнить печатаемое слово	<Ctrl+пробел>

Работа со свойствами

Для того чтобы...	нажмите...
Перейти в окне свойств к следующему свойству, начинающемуся с заданной буквы	<Ctrl+Shift+заданная буква>

Выполнение **программы**

Для того чтобы...	нажмите...
Запустить процедуру или форму в активном окне	<F5>
Приостановить выполнение программного кода и перейти в режим паузы	<Ctrl+Break>
Начать выполнение программного кода с остановкой на строке, содержащей текстовый курсор	<Ctrl+F8>

Управление окнами

Если ваш монитор не отличается гигантскими размерами экрана, приготовьтесь потратить довольно много времени на перемещение окон редактора Visual Basic. Эти окна существуют не столько для того, чтобы на них смотреть, сколько для того, чтобы оказать достаточно существенную помощь в вашей программистской работе. Проблема заключается в том, что держать их все на экране открытыми нерационально — вам совсем не останется места для работы с формами и VBA-кодом.

Одни окна любят одиночество, другие — всегда в компании

Относительно окон редактора Visual Basic вы должны запомнить один основной факт: окон форм и программного кода можно открыть столько, сколько вы пожелаете, а каждое из окон остальных типов может присутствовать на экране только в единственном экземпляре. Вероятно, вам такое ограничение сразу покажется вполне логичным, а вот лично мне, чтобы разобраться, потребовалось некоторое время.

Несколько окон для программного кода и форм можно открыть потому, что обычно создаются несколько форм и один VBA-модуль. Имея для каждого из этих объектов свое окно, вы получите быстрый доступ сразу ко всем своим формам и модулям.

Некоторые другие окна (а конкретно — окно свойств и окно локальных переменных) изменяют свое содержимое автоматически, когда вы переходите в окно другой формы или соответствующего программного кода. Это похоже на несколько окон по цене одного. Другие же окна — например, окно обозревателя объектов, окно проводника проекта и окно немедленного выполнения • — применимы ко всему, что делается в редакторе Visual Basic, поэтому можно открыть только по одному из них.

Видимые и скрытые окна

Почти для каждого окна редактора Visual Basic есть комбинация клавиш, которая открывает его. Это значит, что если окно невидимо, вы можете открыть его, не отрывая рук от клавиатуры. Все эти комбинации клавиш представлены в табл. 5.1. Если вам трудно их запомнить, всегда можно открыть любое из окон через меню View.

Жаль, что эти комбинации клавиш *не* работают, как выключатели. Если окно уже открыто, нажатие соответствующей комбинации клавиш не прячет его. Чтобы заставить окно уйти, вам придется щелкнуть на маленькой кнопке закрытия X, которая расположена в правом верхнем углу окна.

Чтобы открыть окно, которое уже отображено на экране, но затерялось за другими окнами, выберите имя этого окна из меню Window. В этом меню будут представлены только перемещаемые окна.

Как и во многих других приложениях Windows, для переключения от одного окна к другому вы можете использовать две стандартные комбинации клавиш — <Ctrl+Tab> и <Ctrl+F6>. Однако в редакторе Visual Basic в цикле таких переходов будут участвовать только перемещаемые окна.

Вы, наверное, спросите: "Что такое перемещаемые окна?" Ответ на этот вопрос находится в следующем разделе.

Закрепленные и свободно перемещаемые окна

Подобно панелям инструментов, большинство окон редактора Visual Basic бывают *закрепленными*, т.е. вы можете привязать их к любой из четырех сторон рабочей области главного окна, где их не перекрывают другие окна. Ясно, что закрепление окна делает рабочее пространство меньше. На рис. 5.5 показан редактор Visual Basic, в котором закреплены все допускающие закрепление окна.

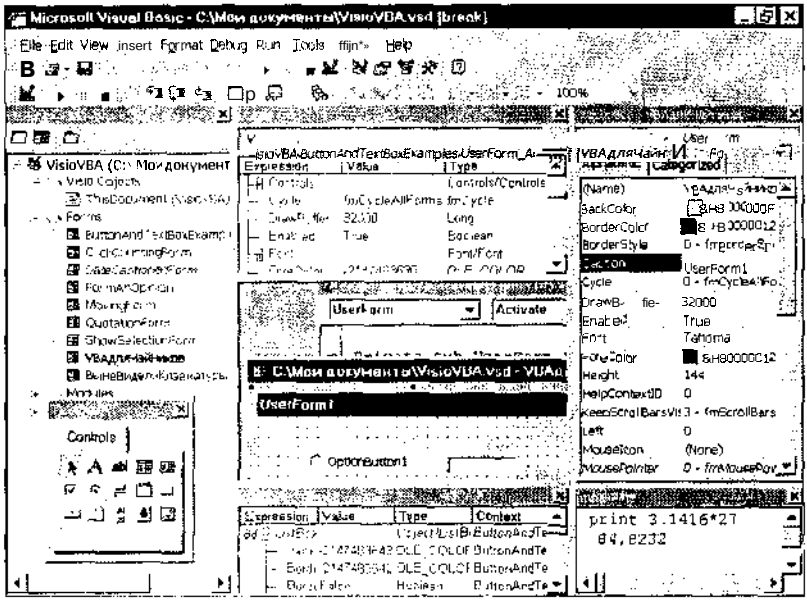


Рис. 5.5. По мере увеличения числа закрепленных окон рабочее пространство редактора Visual Basic, предназначенное для окон редактирования кода и проектирования форм, становится все меньше и меньше

В качестве альтернативы можно сделать окна свободно перемещаемыми по экрану. Перемещаемые окна дают больше пространства для редактирования кода и проектирования форм, но зато часто перекрывают другие окна. Если у вас достаточно большой монитор, наверное, вы предпочтете установить размеры окна редактора Visual Basic меньшими, чем весь экран, и разместить эти перемещаемые окна вне главного окна вообще.

Парковка запрещена!

Есть окна, которые нельзя сделать закрепленными, например окна программного кода и пользовательской формы. Кроме того, любое окно вообще можно сделать *незакрепляемым*. Чтобы проверить или установить состояние способности закрепляться для всех окон (конечно, кроме окон программного кода и пользовательской формы), нужно выбрать сначала Tools⇨Options, а затем в появившемся окне щелкнуть на вкладке Docking. *Я* уверен, что вы доберетесь до этой вкладки и без рис. 5.6, но из-за полной неразберихи на предыдущей иллюстрации вы, наверное, желаете взглянуть на что-нибудь более упорядоченное и понятное.

Можно также сделать закрепляемое окно не способным к закреплению (если это окно в данный момент закреплено), щелкнув правой кнопкой мыши в строке его заголовка, а затем выбрав Dockable в появившемся контекстном меню. К сожалению, в обратном направлении этот прием не работает.

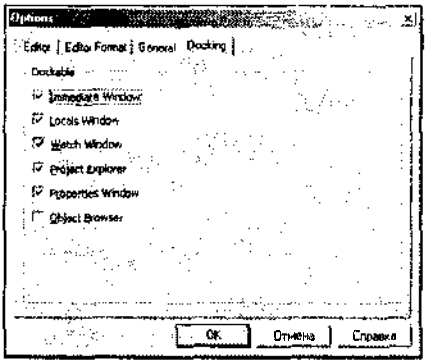


Рис. 5.6. В таком диалоговом окне устанавливается способность к закреплению окон редактора Visual Basic

Несколько закрепляющих фактов

Вот некоторая информация для размышления, пока вы пытаетесь понять принципы закрепления окон.

- ✓ Окна программного кода, пользовательской формы, как и любое другое не способное закрепляться окно, можно развернуть, свернуть или восстановить (последнее на жаргоне *Microsoft* означает вернуть к размеру, определенному пользователем). Не способные закрепляться окна имеют полный комплект стандартных кнопок Windows в правом верхнем углу — это кнопки сворачивания, разворачивания (или восстановления, если окно уже развернуто) и закрытия окна.
- ✓ Можно сразу определить, является ли окно закрепляемым, если посмотреть на его полосу заголовка. Закрепляемые окна имеют только кнопку закрытия окна, а не способные закрепляться окна имеют также кнопки свертывания и развертывания.
- ✓ Если развернуть не способное к закреплению окно, то оно займет все свободное пространство рабочей области, оставленное закрепленными окнами.
- ✓ Чтобы иметь возможность разместить перемещаемое окно возле края рабочей области, избежав при этом закрепления окна, сначала нужно отключить возможность закрепления для этого окна.
- ✓ Все открытые и не способные к закреплению окна доступны последовательно с помощью комбинации клавиш <Ctrl+Tab>. Эти окна перечислены также в меню Window.

Сохранение структуры экрана

Редактор Visual Basic автоматически сохраняет текущую структуру экрана при выходе из него. Это значит, что при следующем входе в редактор Visual Basic все ваши окна, меню и панели инструментов будут на тех же самых местах, где их в прошлый раз покинули.

Управление проектами с помощью проводника проектов

В VBA термин *проект* используется для обозначения программного кода и всех форм, принадлежащих одному документу, в совокупности с самим документом. В редакторе Visual Basic для панорамного обзора всех проектов, открытых в приложении, и, что еще важнее, для быстрого доступа к окнам программного кода и пользовательских форм используется окно Project Explorer (Окно проводника проектов).

Чтобы сохранить проект, с которым вы работаете в данный момента (а как объясняется ниже, это именно тот проект, который выделен в окне проводника проектов), щелкните на кнопке Save (Сохранить) в панели инструментов Standard (Стандартная) редактора Visual Basic. При этом документ сохраняется вместе с ассоциированным с этим документом программным кодом, а значит, для сохранения документа совсем не обязательно возвращаться в VBA-приложение.

Вызов проводника проектов

Окно проводника проектов должно присутствовать на экране, когда вы впервые открываете редактор Visual Basic, Если же это окно отсутствует, то, для того чтобы сделать его видимым, необходимо выполнить любое из следующих действий.

- ✓ Нажать комбинацию клавиш <Ctrl+R>.
- ✓ Щелкнуть на кнопке Project Explorer на панели Standard.
- ✓ Выбрать команду View⇨Project Explorer.

Исследование проводника проектов

Если вы когда-нибудь пользовались проводником Windows для работы со своими дисками и файлами (ой, извините, *документами),* то и в окне проводника проектов вы будете чувствовать себя как дома. Проводник проектов предлагает вам иерархическое представление открытых проектов в виде древовидной структуры (рис. 5.7).

Наверху иерархической структуры находятся сами проекты, их строки придвинуты к левому краю окна проводника проектов ближе других.

Каждый из открытых в приложении документов автоматически представляется как проект, даже если вы еще не вводили для этого проекта никакого программного кола и не создавали форм. По умолчанию проект получает имя породившего его документа, но можно изменить имя проекта в диалоговом окне Project Properties (Свойства проекта) или в окне свойств (ознакомьтесь ниже с разделом "Изменение имени проекта или модуля в окне свойств").

Следующую ступень в иерархии занимают группы объектов — формы, модули программного кода, объекты содержащего документ приложения, а также ссылки на другие библиотеки объектов (о том, как добавить в проект такие ссылки, говорится в главе 14). Обычно каждая из таких групп в окне проводника проектов представлена как папка. Сами же отдельные объекты образуют последний уровень в этой иерархии.

Перемещение в окне проводника проектов

Бесцельное движение по дереву проектов в окне проводника проектов можно использовать, чтобы выглядеть очень занятым, не создавая при этом никакого программного кода. Но проводник проектов полезен не только этим — он обеспечивает самый быстрый способ нахождения и активизации модулей, форм и других объектов, с которыми вы собираетесь работать.

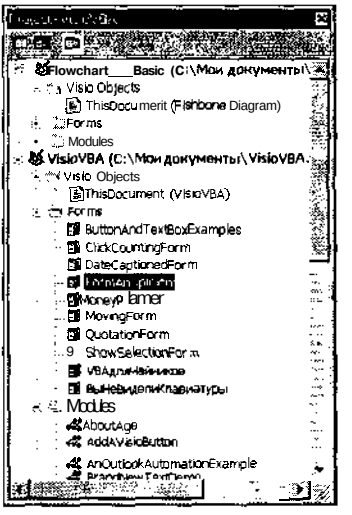


Рис. 5.7. Пример окна проводника проектов в Visio с несколькими одновременно открытыми проектами

Подобно проводнику Windows, проводник проектов в Visual Basic отображает в своем списке небольшой квадратик около каждой группы (ее обычно называют *разделом).* Когда видимо только имя проекта, а его содержимое скрыто, этот квадратик, называемый *индикатором развертывания,* содержит знак "плюс". Если же видны разделы следующего уровня, то этот индикатор показывает знак "минус".

При работе с проводником проектов используются следующие приемы.

- ✓ Чтобы развернуть раздел и показать объекты следующих уровней иерархии, щелкните на индикаторе развертывания, когда последний содержит знак "плюс", или выделите (подсветите) раздел и нажмите клавишу со стрелкой, направленной вправо.
- ✓ Чтобы свернуть уже развернутый раздел и скрыть его дочерние объекты, щелкните на индикаторе развертывания, когда он содержит знак "минус", или выделите раздел и нажмите клавишу со стрелкой, направленной влево.
- ✓ Чтобы открыть окно пользовательской формы или окно программного кода для данного раздела модуля, формы или модуля класса, дважды щелкните в строке соответствующего раздела в списке или выделите раздел и нажмите <Enter>.
- ✓ Чтобы вызвать окно программного кода пользовательской формы, выделите эту форму в списке и нажмите <Shift+Enter>.

Если вам трудно запомнить, когда следует использовать <Enter>, а когда <Shift+Enter>, используйте кнопки в верхней части окна проводника проектов. Они работают следующим образом.

- ✓ Кнопка View Code (Показать программный код), расположенная слева, отображает на экране окно программного кода для выделенного объекта.
- ✓ Кнопка View Object (Показать объект), расположенная посередине, отображает на экране сам выделенный объект. Если этот объект— форма, вы увидите ее в окне пользовательской формы. Если это документ, вы автоматически переключитесь в окно вызвавшего редактор Visual Basic приложения, и в этом окне активным будет выделенный вами документ.

✓ Кнопка Toggle Folders (Выключатель отображения папок), расположенная справа, включает или отключает отображение среднего уровня иерархии. В обычном режиме проводник проектов сортирует формы, модули программного кода и объекты документа по отдельным папкам. Если щелкнуть на этой кнопке, соответствующие папки пропадут с экрана и все объекты проекта предстанут на экране в едином списке, упорядоченном по алфавиту. Чтобы вернуть папки, нужно щелкнуть на этой кнопке еще раз.

Использование контекстного меню проводника проектов

Контекстное меню проводника проектов (рис. 5.8) предоставляет дополнительные возможности для удобной работы с элементами проекта. Все команды этого контекстного меню доступны и через главное меню, но ясно, что во время работы с проводником проектов контекстное меню обеспечивает существенно более быстрый доступ к этим командам.

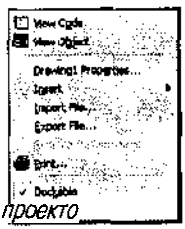


Рис. 5.8. Контекстное меню проводника проекто

Установка свойств проекта

В диалоговом окне Project Properties (Свойства проекта) можно изменить имя проекта, добавить его краткое описание, назначить проекту файл справки, а также защитить проект от несанкционированного любопытства и изменений. Правда, пока вы не станете достаточно опытным разработчиком VBA-программ, большинство из этих возможностей вам будут не слишком полезны.

И все же вы, по крайней мере, уже знаете, что такое диалоговое окно существует. Чтобы отобразить его на экране, выберите Tools⇨Имя проекта Properties из меню или щелкните в окне проводника проектов правой кнопкой мыши и выберите Имя проекта Properties из контекстного меню. На рис. 5.9 вы видите диалоговое окно Project Properties с открытой вкладкой General (Общие).

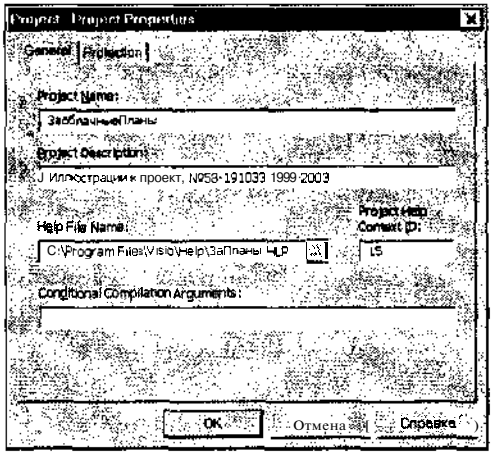


Рис. 5.9. Вкладка General диалогового окна Project Properties

Переименованиепроекта

В списках проводника проектов и обозревателя объектов проекты упорядочены по алфавиту, а это значит, что для перемещения проекта в списке этот проект нужно переименовать. (Конечно, вы можете изменить имя проекта и из чисто эстетических соображений.)

Проще всего изменить предложенное VBA имя проекта с помощью окна Properties (Окно свойств). Когда вы выделяете какой-либо проект в окне проводника проектов, имя этого проекта появляется в окне свойств. В окне свойств это имя можно изменить на любое другое по своему усмотрению. Изменить имя проекта можно и на вкладке General диалогового окна Project Properties, если это диалоговое окно по каким-либо причинам оказалось открытым.

Защитапроекта

Если есть вероятность того, что кто-то может получить доступ к вашему компьютеру и что-нибудь сделать с вашими программами, подумайте о защите своих программ паролями. Чтобы возвести заграждения своей безопасности, откройте диалоговое окно Project Properties и перейдите на вкладку Protection (рис. 5.10).

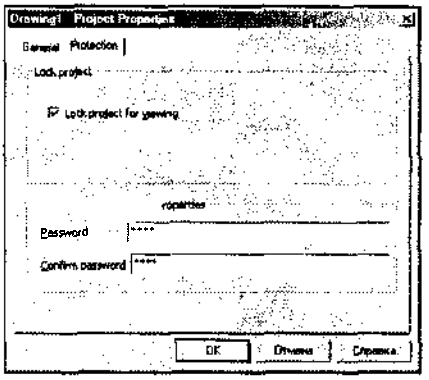


Рис. 5.10. ВкладкаProtectionдиалоговогоокна ProjectProperties

Вот как пользоваться возможностями, предлагаемыми на этой вкладке.

- ✓ Lock project for viewing (Закрыть проект для просмотра). Установка флажка этой опции полностью лишает любого, кто не знает пароля, возможности даже увидеть проект. Если этот флажок установлен, пароль не даст другим открыть диалоговое окно Project Properties, а значит, не даст никому другому возможности закрыть для вас вашу собственную работу.
- ✓ Password (Пароль). Напечатайте здесь пароль, который вы выбрали для данного проекта.
- ✓ Confirm Password (Подтвердите пароль). Здесь напечатайте пароль еще раз для гарантии, что при его вводе в предыдущее поле вы не допустили ошибки.

Если ваша память неидеальна, запишите этот пароль и сохраните его там, где вы сможете потом его найти. VBA достаточно надежно зашифровывает пароль, и если вы его потеряете, вам придется создавать проект вновь с самого начала.

использованиеобозревателя объектов

Окно Object Browser (Обозреватель объектов) выглядит посложнее, но мало чем отличается от окна проводника проектов. Подобно проводнику проектов, обозреватель объектов предоставляет возможность быстрого доступа к объектам, доступным вашей VBA-программе.

Не считая "косметических", главным отличием обозревателя объектов является то, что он показывает только один проект— но зато предлагает вам возможность добраться ко всем объектам, доступным для этого проекта, а не только к тем, которые принадлежат самому проекту. Другими словами, кроме модулей и форм своего проекта, вы можете увидеть и объекты, предлагаемые вашим приложением, VBA, а также другими библиотеками объектов, которые могли быть открыты для использования проектом (инструкции по поводу добавления ссылок на другие библиотеки объектов приводятся в главе 14).

Другое важное преимущество обозревателя объектов в том, что с его помощью можно ознакомиться с процедурами, методами, событиями, свойствами и другими подобными штуками любой такой библиотеки объектов.

Вызов обозревателя объектов

Самый быстрый способ вызова окна обозревателя объектов — нажатие клавиши <F2>. Медленнее к той же цели ведут щелчок на кнопке Object Browser в панели инструментов Standard и выбор View⇨Object Browser из меню. Пример окна обозревателя объектов показан на рис. 5.11.

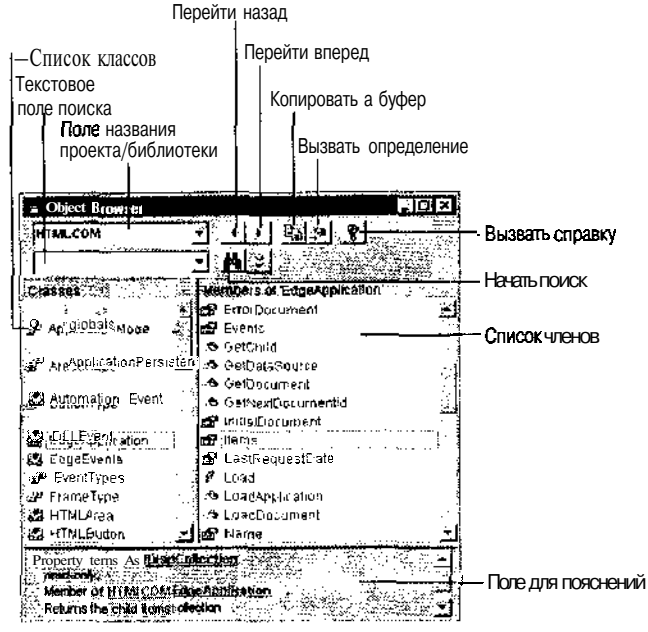


Рис. 5.1 /. Окнообозревателяобъектов

Просмотр объектов

Прежде чем выполнять любые серьезные действия с помощью обозревателя объектов, убедитесь, что вы работаете с нужным проектом. Для этого необходимо только выбрать подходящий проект в окне проводника проектов.

Щелчок в поле раскрывающегося списка Project/Library (Список проектов/библиотек) слева вверху окна открывает список библиотек, доступных данному проекту. (В том же списке должен присутствовать и сам проект, так что если его нет — возвратитесь в окно проводника проекта и убедитесь, что нужный проект действительно выбран.) Выбрав в списке <All Libraries> (Все библиотеки), вы сможете просмотреть все доступные в проекте объекты, т.е. объекты из всех библиотек, на которые ссылается проект. Чтобы снова сузить область просмотра до одной библиотеки, выберите ее из этого списка.

Обзор панелей обозревателя

Окно обозревателя объектов разделено на несколько панелей (см. рис. 5.11). Размеры любой из этих панелей можно изменить, перетаскивая разделительные линии между ними. В верхней панели есть кнопки, с помощью которых можно управлять обозревателем объектов. Ниже расположена панель для вывода результатов поиска; обсуждение этой панели будет представлено ниже, в разделе "Поиск членов".

Ниже рядом расположены еще две панели. Левая содержит список Classes (Список классов), в котором показаны объекты, классы, коллекции, модули, формы и константы из выбранной библиотеки или проекта. Щелчок в любой из строк этого списка открывает список Member of (Список членов) соответствующего объекта в правой панели. Этими членами могут быть процедуры, методы, свойства, события или отдельные константы.

Желаете знать, где же прописался ваш собственный программный код? А вы посмотрите повнимательнее на те объекты в списке классов и списке членов, которые отображаются полужирным шрифтом.

Панель Details (Подробности) в самом низу окна обозревателя объектов отображает информацию в выбранном в данный момент объекте, в частности его тип и содержащий его объект (если вы не видите всю эту информацию из-за слишком маленьких размеров панели, перетащите верхний разделитель панели выше или используйте вертикальную полосу прокрутки в правой стороне панели). Щелчок на объекте, содержащем данный, заставит обозреватель объектов перейти к рассмотрению этого объекта.

Движение по объектам

Работа с обозревателем объектов очень напоминает работу с обозревателем Web-страниц. Как и в обозревателе Web-страниц, здесь вы тоже можете использовать гиперссылки для перехода к родственным объектам — щелчок на подчеркнутом тексте в панели Details (Внизу окна) отобразит соответствующий объект в окне обозревателя объектов.

Немедленный доступ к программному коду

Если объект, выбранный вами в окне обозревателя объектов, является модулем или процедурой из вашей собственной программы, просто нажмите <Enter> — и вы увидите окно Code с соответствующим программным кодом. Любители упражнений с мышью могут вместо этого щелкнуть на кнопке Show Definition (Показать определение).

Получение справки из окна обозревателя объектов

Предполагается, что нажатие клавиши <F1> или щелчок на кнопке Help (Справка) вверху окна обозревателя объектов должен открыть раздел справки, соответствующий объекту, методу, свойству или событию, которые оказались выделенными в окне обозревателя объектов. Обычно так и бывает. Если ваш стиль мышления не слишком регламентирован, то случайное блуждание в окне обозревателя объектов и вызов справки для заинтересовавших вас при этом объектов могут оказаться прекрасным способом безболезненного освоения VBA.

Поиск членов

Предположим, вы не помните, какой из модулей программного кода содержит нужную вам процедуру или какой из объектов имеет определенный метод или событие. Вместо того чтобы в этом случае копаться в своих модулях с помощью окна проводника проекта или вызывать справочную систему, можно заставить обозреватель объектов найти для вас соответствующий объект. Вот как это сделать.

1. **В раскрывающемся списке библиотек/проектов выберите <All Libraries> (если вы собираетесь искать во всем проекте) или соответствующую библиотеку.**
   Если вы ошиблись в выборе библиотеки, может случиться, что вы обнаружите объект с нужным именем, но этот объект окажется совсем не тем, который нужен вам, поскольку объекты в разных библиотеках могут иметь совпадающие имена.
2. **В поле Search (Найти) введите текст, который требуется найти.**
   Можно повторить поиск любой из четырех введенных последними текстовых строк, выбрав подходящую из раскрывающегося списка поля **Search.**
3. **Нажмите <Enter> или щелкните на кнопке Search (Найти), на которой изображен бинокль.**
   После обязательного при этом дребезжания жесткого диска вы увидите удовлетворяющие условию поиска объекты в новой, специально появившейся для этого панели **Search Results** (Результаты поиска) над панелями со списками. Здесь тоже можно изменить размеры любого из столбцов, если они слишком малы для отображения нужных объектов.
4. **Чтобы закрыть панель Search Results, щелкните на кнопке с двойной направленной вверх стрелкой рядом с кнопкой Search.**
   Если щелкнуть на той же кнопке еще раз (но теперь на ней будет изображена двойная стрелка, направленная вниз), панель Search Results появится снова.

Использование информации из окна обозревателя объектов в программном коде

Обозреватель объектов удовлетворит любое любопытство к объектам в проекте, но, кроме того, он может служить и довольно неплохим средством создания программного кода. После того как вы найдете что-либо, что хотели бы использовать в своей программе, нажмите <Ctrl+C> или щелкните на кнопке Copy to Clipboard (Копировать в буфер), чтобы поместить найденное в буфер обмена. После этого переключитесь в окно программного кода и вставьте содержимое буфера обмена в свою программу. Такой метод гарантирует, по крайней мере, отсутствие синтаксических ошибок — вы же знаете, что даже самая ничтожная опечатка может свести с ума любую программу (другими словами, программа никогда не выполняется правильно с первого раза).

Секреты кодирования

Окна Code (Окна программного кода) — это сердце редактора Visual Basic: в них вы создаете VBA-операторы, непосредственно выполняющие полезную работу. В этой главе я не собираюсь говорить об использовании VBA-операторов, сосредоточусь на том, как получить максимум пользы от окна программного кода при создании программы. В VBA имеется три типа объектов, с которыми ассоциированы окна программного кода, — это модули, модули классов и пользовательские формы. И хотя все эти объекты выполняют в VBA разные функции, которые обсуждаются в последующих главах, их окна программного кода выглядят и работают совершенно аналогично.

Вызов окна программного кода

Для того чтобы открыть окно программного кода для существующего модуля, модуля класса или пользовательской формы, в редакторе Visual Basic предусмотрен целый ряд совершенно различных способов. Сначала нужно найти и выделить объект в окне проводника проектов. После этого для вызова соответствующего окна программного кода можно выполнить любое из следующих действий.

- ✓ Нажать клавишу <F7>.
- ✓ Щелкнуть на кнопке View Code в верхней части окна проводника проектов.
- ✓ Щелкнуть на объекте правой кнопкой мыши и выбрать View Code из появившегося контекстного меню.
- ✓ Выбрать View⇨Code.
- ✓ В случае модуля (но не пользовательской формы!) дважды щелкнуть на нем или нажать клавишу <Enter>.

Если вы находитесь в окне пользовательской формы, то нажатие клавиши <F7> или выбор View⇨Code откроет окно программного кода этой пользовательской формы.

Создание нового окна программного кода

При добавлении в проект нового модуля автоматически открывается новое окно пользовательского кода для этого модуля. (Процесс добавления новых модулей обсуждается в главе 6.) При создании новой формы для нее автоматически создается и окно программного кода (но вы не увидите его, пока не выполните действия, описанные в предыдущем разделе).

Печатание программного кода

Окна программного кода в VBA являются, по сути, простыми текстовыми редакторами, правда, у них есть специальные возможности для создания и редактирования именно VBA-кода.

Печатание VBA-операторов происходит точно так же, как и в обычном текстовом процессоре, с использованием тех же приемов работы с текстовым курсором и клавиш редактирования текста, стандартных для Windows (нажатие клавиши <Home> осуществляет переход в начало строки, а <Ctrl+Home> — в начало окна). Точно так же можно выделять текст либо с помощью мыши, либо нажав и удерживая клавишу <Shift> при перемещении курсора.

Как и любой уважающий себя текстовый редактор начала нового тысячелетия, окно программного кода поддерживает возможность перетаскивания выделенного текста. После того как текст, с которым вы хотели бы работать, выделен, вы можете сделать следующее.

- ✓ Переместить выделенный текст, перетащив его в другое место и опустив там.
- ✓ Скопировать выделенный текст в другое место, действуя так же, как и при перемещении, но с нажатой клавишей <Ctrl>.

Можно перетаскивать текст в новую позицию в том же окне, в другое окно программного кода, а также в окна Immediate (Окно немедленного выполнения команд) и Watches (Окно контролируемых выражений). Если пунктом назначения является другое окно программного кода, то перед тем, как начать перемещение, нужно разместить окна так, чтобы и исходный текст, и место назначения были видны на экране. (Краткое описание окна немедленного выполнения команд и окна контролируемых выражений вы найдете ниже, в разделе "Краткое знакомство с окнами для отладки".)

Окно программного кода позволяет отменить последние из внесенных изменений в программный код. Каждый раз, когда вы нажмете <Ctrl+Z> или выберете Undo (Отменить ввод) из меню Edit (Правка), будет отменено следующее из изменений. Меню Edit предлагает и команду Redo (Повторить ввод) — отмену применения команды Undo, — но команде Redo в редакторе Visual Basic не назначено никакой комбинации клавиш для быстрого ее вызова с клавиатуры.

Идеальный тренер

Как надежный слуга, редактор Visual Basic постоянно (но ненавязчиво) проверяет и подправляет вашу работу следующим образом.

- ✓ Если вы напечатаете одну строку программного кода с отступом, тот же отступ автоматически будет установлен и для следующих строк (это можно отключить, выбрав сначала Tools⇨Options из меню, а затем в появившемся диалоговом окне сняв флажок Auto Indent (Автоматический отступ)).
- ✓ Если редактор Visual Basic распознает *ключевое слово,* он автоматически перепишет его с прописной буквы в соответствии с соглашениями VBA (например, если вы напечатали `if...then...else`, редактор превратит это в `If...Then...Else`). Кроме того, ключевые слова автоматически выделяются цветом (по умолчанию — синим), чтобы они были видны на фоне других слов.
- ✓ Когда в окне программного кода, создавая процедуру, вы вводите ключевое слово `Sub` (или `Function`), за которым следуют скобки со списком аргументов внутри них, редактор Visual Basic автоматически дописывает за вас необходимый в этом случае оператор `End Sub` (или `End Function`). Обратите внимание, что между процедурами вставляется также строка-разделитель.
- ✓ Наконец, что еще важнее, если напечатать VBA-оператор, который явно не полон или каким-либо другим образом не согласуется с синтаксисом языка, редактор Visual Basic отобразит на экране соответствующее сообщение (рис. 5.12). Благодаря такому немедленному замечанию вы можете исправить ошибку, чтобы потом не вспоминать, как вы собирались дополнить этот оператор. Если же вы игнорируете предупреждение, редактор Visual Basic изменит цвет оператора на красный, чтобы напоминать о том, что в данном операторе что-то не так.

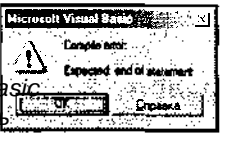


Рис. 5.12. Если строка программного кода не отвечает установленным правилам синтаксиса, редактор Visual Basic выводит на экран сообщение, подобное показанному здесь

Перемещение в окне программного кода

В программном коде практически любой программы может оказаться немало строк, и тогда простое перелистывание текста программы в окне программного кода для поиска будет выглядеть слишком примитивным в таком элегантном окружении. Куда лучше воспользоваться возможностями, предоставляемыми двумя раскрывающимися списками вверху окна программного кода (рис. 5.13). Они могут перенести вас прямо к той процедуре, которую вы хотели видеть или редактировать. Вот что могут эти раскрывающиеся списки.

- ✓ Раскрывающийся список слева — это *список объектов*. В окне программного кода модуля здесь присутствует только пункт (General) (Общие), так что в случае модуля на этот список можно не обращать внимание вообще. Другое дело, когда вы работаете в окне программного кода для формы. В этом случае из списка объектов вы можете выбрать конкретный помещенный в форму элемент управления (или саму форму). В результате такого выбора в окне профаммного кода будет показана процедура, являющаяся для соответствующего объекта процедурой по умолчанию.
- ✓ Раскрывающийся список справа — это *список процедур/событий*. Здесь можно выбрать либо раздел (Declarations) (Объявления) для всего окна, либо конкретную процедуру, чтобы отобразить в окне соответствующий программный код. Если вы находитесь в окне программного кода формы, этот список будет содержать только события, допустимые для объекта, выбранного в списке объектов. При выборе события в окне программного кода появляется процедура, соответствующая этому событию.

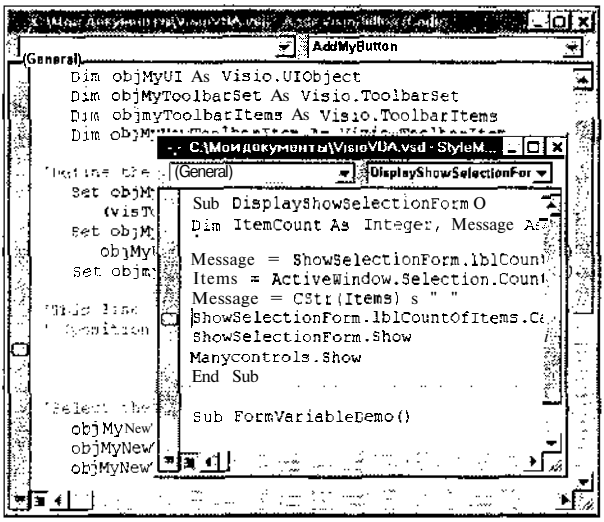


Рис. 5.13. ЭТИ светлые овалы на полосе в левой части окна программного кода представляют закладки. По умолчанию на цветном мониторе они голубого цвета

Закладки в программном коде

Предположим, сейчас 3 часа ночи. Ваши глаза слипаются, и вас клонит ко сну, но сроки нещадно поджимают, и вы упорно отбиваете на клавиатуре строку за строкой довольно однообразный программный код. Вдруг — прилив вдохновения — вы понимаете, как можно решить одну важную программистскую проблему, над решением которой вы перед этим уже безрезультатно бились добрую половину дня.

Прежде чем перейти в тот модуль, в котором будет воплощена ваша гениальная идея, поместите *закладку* в том месте, где вы работаете в данный момент. И когда наступит время вернуться, вы без труда очень быстро отыщете дорогу назад, подобно Гензелю и Гретель,

Чтобы поместить закладку в строку программного кода (точнее, рядом со строкой программного кода; рис. 5.13), щелкните на кнопке Toggle Bookmark (Добавить закладку). Все относящиеся к закладкам кнопки находятся в панели инструментов Edit (Правка), поэтому, чтобы получить доступ к ним, нужно отобразить эту панель инструментов на экране. Иначе можно использовать соответствующую команду меню (Edit⇨Bookmarks⇨ Toggle Bookmark), но это существенно более длинный путь. (Это жестоко, но *Microsoft* не предусмотрела комбинации клавиш для закладки.) Закладок можно разместить столько, сколько вы пожелаете.

Ясно, что сама по себе закладка не принесет никакой пользы. Щелкайте на кнопках Next Bookmark (Следующая закладка) и Previous Bookmark (Предыдущая закладка) для последовательного перехода от закладки к закладке, пока не доберетесь до места назначения.

Чтобы удалить отдельную закладку, поместите текстовый курсор в строку с этой закладкой и снова щелкните на кнопке Toggle Bookmark. Если же закладок набралось столько, что использование команд перехода от закладки к закладке уже почти эквивалентно использованию полосы прокрутки, удалите все закладки сразу одним щелчком на кнопке Clear All Bookmarks (Удалить все закладки).

Разделение окна программного кода

Любое окно программного кода можно разделить на два (рис. 5.14). Это позволит вам видеть одновременно различные части программного кода одного и того же модуля. В этом случае очень удобно также копировать текст из одной части программного кода в другую.

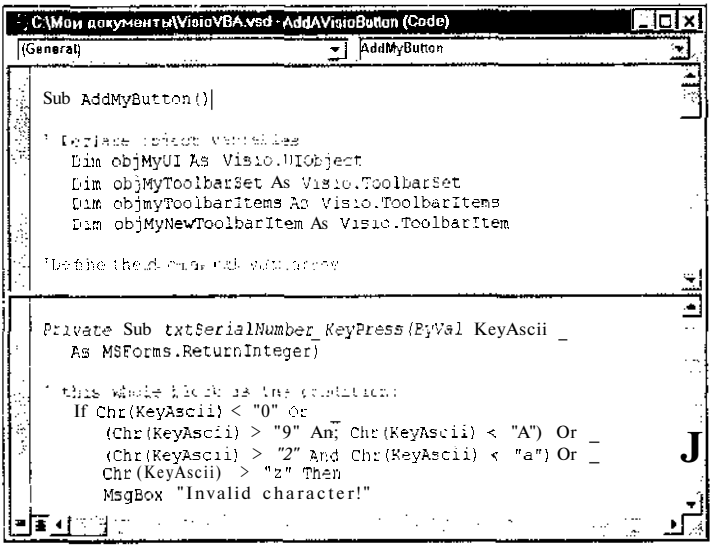


Рис. 5.14. Я разделил этоокнопрограммногокода

Чтобы разделить окно программного кода, используйте линию разбивки — маленький серый бегунок сразу над верхней стрелкой вертикальной полосы прокрутки в правой части окна. Перетащите линию разбивки вниз, установив подходящие размеры частей окна. Чтобы вновь сделать окно единым, уберите линию разделения двойным щелчком на ней или перетащите ее снова на самый верх окна.

Созидательные возможности окна программного кода

Зачем печатать программный код самим, если кто-то может напечатать его за вас? Редактор Visual Basic балует вас возможностью автоматического ввода совершенно непригодных для запоминания терминов VBA и как раз в самый подходящий момент. Это не только минимизирует печатную работу, но и гарантирует отсутствие опечаток.

Все эти любезности обеспечиваются несколькими замечательными средствами редактора Visual Basic, команды вызова которых помещены в меню Edit (Правка).

- ✓ List Properties and Methods (Получить список свойств и методов).
- ✓ List Constants (Получить список констант).
- ✓ Complete Word (Автоматически дополнить слово).

Использование списка свойств и методов

Изо всех средств, о которых идет здесь речь, средство List Properties and Methods (Получить список свойств и методов) будет, пожалуй, самым полезным. Вот как этим средством пользоваться. Чтобы сделать что-либо с VBA-объектом, обычно меняют одно из его свойств или вызывают один из его методов. Чтобы идентифицировать свойство или метод, с которым необходимо работать, печатают сначала имя объекта, затем точку (оператор *точка),* имя свойства или метода. Например:

```
ActiveWindow.Selection.Group
```

(Подробное описание приемов работы с объектами и их свойствами и методами вы найдете в главе 12.)

Чтобы воспользоваться средством вызова списка свойств и методов, вам нужно только ввести имя объекта и точку. Как только вы это напечатаете, на экране появится небольшое окно со списком всех присущих этому объекту свойств и методов (рис. 5.15). Чтобы найти подходящее свойство или подходящий метод, прокрутите список (с помощью клавиш со стрелками) или напечатайте одну-две первые буквы имени, После того как подходящее имя будет выделено, нажмите <Tab>, чтобы вставить это имя в свой программный код.

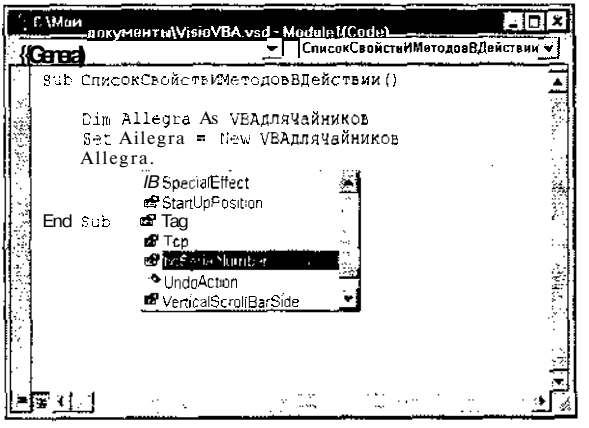


Рис. 5.15. Средство List Properties and Methods очень полезно при вводе в программный код имен свойствиметодов

Аналогично это средство работает и при объявлении переменной, только в данном случае возникает список допустимых типов. Например, если начать печатать оператор

```
Dim Oatmeal As CerealObject
```

то, сразу после того как будет напечатано слово As, появится список всех имеющихся типов данных и объектов. Как и выше, можно выбрать нужный пункт из списка, пролистав список или напечатав пару букв, а затем нажав клавишу <Tab>. Кстати, целый поток информации по поводу объявления переменных ждет вас в главе 7.

Средство вызова списка свойств и методов работает не только для встроенных объектов VBA, но и для определенных вами переменных — *за исключением тех, которые имеют тип Variant.* Между прочим, это еще одна причина, по которой стоит явно определять переменные. (Подробно об этом — в главе 7.)

Да, и еще один момент. Список свойств и методов появляется автоматически только тогда, когда эта возможность активизирована в диалоговом окне Options (Параметры). Если у вас список не появляется, выберите Tools⇨Options и проверьте, отмечен ли флажок Auto List Members (Автоматический вызов списка членов). Этот список можно вызвать и соответствующей командой из меню Edit, и с помощью клавиш <Ctrl+J>.

Использованиесписказначений

Средство List Constants (Получить список констант) работает практически так же, как и средство получения списка свойств и методов, конечно, за исключением того, что в данном случае отображается список констант, определенных для свойства, с которым вы имеете дело. Этот список появляется сразу же после того, как вы напечатаете знак =, например, в строке оператора

```
Oatmeal.Texture = Gluey
```

Лично я не могу понять, почему это средство имеет отдельную команду, а не рассматривается как часть средства List Properties and Methods (Получить список свойств и методов). Но как есть, так и есть, и поэтому для вызова списка констант служит несколько иная комбинация клавиш — <Ctrl+Shift+J>. А чтобы узнать, как использовать сами константы, загляните в раздел "Работа с постоянными значениями" в главе 7.

Автоматическоедополнениеслов

Средство Complete Word (Автоматически дополнить слово) редактора Visual Basic позволяет автоматически ввести практически любой VBA-термин. Активизируется это средство нажатием <Ctrl+пробел>. В результате на экране возникает список, наподобие обсуждавшегося выше списка свойств и методов. Но в данном случае в списке будет практически все, что вы могли бы напечатать, — объекты, функции, процедуры, константы, методы, свойства и созданные вами переменные. Исключение составляют лишь ключевые слова для обозначения встроенных типов данных, таких как Integer и Variant.

Если перед тем, как нажать <Ctrl+пробел>, напечатать одну-две буквы, появившийся список будет значительно короче, поскольку будет содержать только строки, начинающиеся с заданных символов. Более того, если вы напечатаете достаточно начальных символов для того, чтобы в списке осталось только одно слово, редактор Visual Basic дополнит начатое вами, как только будут нажаты клавиши <Ctrl+пробел>, — и без появления какого бы то ни было списка.

Получение списка аргументов

Многие VBA-процедуры требуют, чтобы при их выполнении были определены один или несколько аргументов. Такие процедуры используют в своих вычислениях (или других выполняемых ими действиях) информацию, содержащуюся в этих аргументах. Подобно процедурам, многие методы объектов также требуют задания аргументов.

Тут уж редактор Visual Basic не напечатает аргументы за вас — он просто не может этого сделать, поскольку не знает, какие конкретные значения должны быть у аргументов. Однако он может отобразить на экране что-то вроде шпаргалки — небольшую всплывающую подсказку со списком всех аргументов нужной функции, с указанием типа данных, требуемого каждым из аргументов, и информацией о том, какие из аргументов необязательны. Это всплывающая подсказка Quick Info (Краткая справка), пример ее появления показан на рис. 5.16.

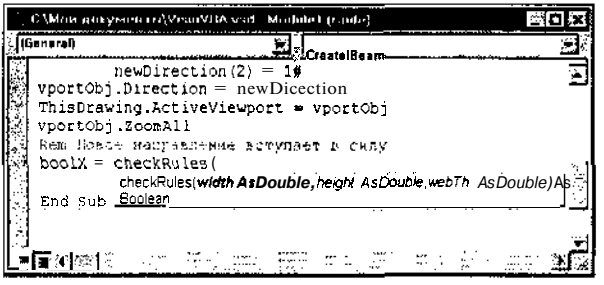


Рис. 5.16. Всплывающая подсказка Quick Info в действии

Если в диалоговом окне Tools⇨Options отмечен флажок **Auto** Quick Info (Автоматический вызов краткой справки), всплывающая подсказка Quick info будет неизменно появляться, как только вы напечатаете имя функции, метода или процедуры, для которых требуется указать аргументы. Если же соответствующая возможность отключена, всплывающую подсказку Quick Info можно вызвать нажатием клавиш <Ctrl+I>.

Существует очень много функций и процедур, которым необходимо указывать несколько аргументов. Чтобы избавить вас от необходимости помнить, где именно в списке аргументов вы находитесь, в подсказке Quick Info печатаемый вами аргумент выделяется полужирным шрифтом. Как только будет напечатана запятая, говорящая об окончании работы с данным аргументом, выделение полужирным шрифтом в окне подсказки перейдет с этого аргумента на следующий.

Quick Info против Parameter Info

При создании достаточно сложных программ хорошим дополнением к подсказкам Quick Info оказываются подсказки Parameter Info (Справка о параметрах). Часто значение аргумента определяется как значение некоторой функции, например:

```
MsgBox(Str(AnIntegerVariable))
```

Здесь функция `Str` конвертирует целое значение, хранящееся в переменной `AnIntegerVariable`, в текстовую строку. Эта строка задает аргумент функции `MsgBox` (и будет сообщением в том окне сообщения, которое VBA отобразит на экране).

Из-за подобного вложения одних функций в другие подсказки Quick Info может оказаться недостаточно, поскольку Quick Info всегда показывает список аргументов той функции, которая содержит точку ввода. Дополнительный же помощник, подсказка Parameter Info, показывает список аргументов для *самой внешней* функции, т.е. для той, в которой содержатся все вложенные. Вызвать подсказку Parameter Info можно с помощью клавиш <Ctrl+Shift+I>.

использованиеокнасвойств

Окно Properties (Окно свойств) позволяет просмотреть и. если нужно, изменить свойства любого объекта (проекта, модуля, формы или элемента управления), который в окне редактора Visual Basic в данный момент активен. Если взглянуть на рис. 5.17, то можно увидеть, что в строке заголовка окна свойств приводится имя активного объекта. В то же время имя этого объекта будет выделено в окне проводника проектов (конечно, если активен элемент управления в форме, то выделено будет имя соответствующей формы, поскольку индивидуальные •элементы управления в окне проводника проектов не отображаются).

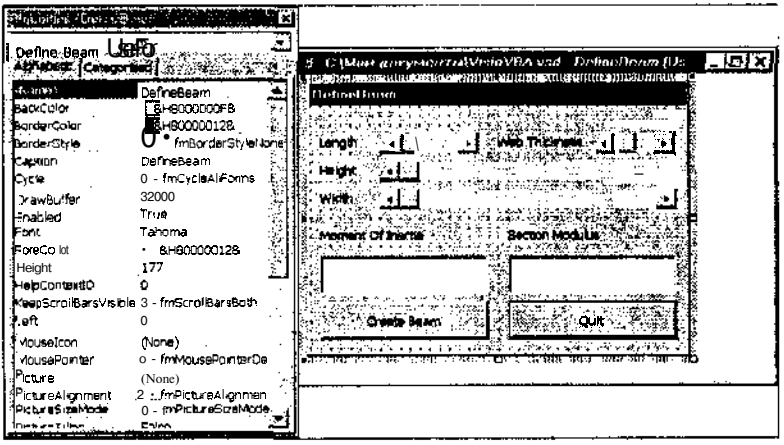


Рис. 5.17. Пример окна свойств для формы

Когда выделен проект или модуль, единственным свойством в окне свойств будет имя этого объекта. Но для формы и элементов управления окно свойств предложит вам мириады свойств, с помощью которых можно управлять внешним видом и поведением данного объекта.

Поскольку окно свойств оказывается исключительно важным главным образом при создании форм, я отложу его обсуждение до главы 10. Здесь же я объясню только, как отобразить это окно на экране и как с его помощью переименовать проект или модуль.

Вызов окна свойств

Чтобы отобразить окно свойств на экране, выполните любое из следующих действий.

- ✓ Нажмите <F4>.
- ✓ Щелкните на кнопке Properties (Свойства) в панели инструментов Standard (Стандартная).
- ✓ Выберите View⇨Properties из меню.

После того как окно свойств появится на экране, перейдите в окно программного кода или в окно пользовательской формы (или щелкните на имени объекта в окне проводника проектов), чтобы свойства этого объекта отобразились в окне свойств.

Изменение имени проекта или модуля в окне свойств

Проекты и модули имеют только одно свойство — имя. С помощью окна свойств вы можете изменить это единственное свойство.

Чтобы переименовать проект или модуль, выполните следующее.

1. **Выделите проект или модуль в окне проводника проектов.**
2. **Перейдите в окно свойств.**

   Если его на экране нет, вызовите его любым из способов, предложенных в разделе "Вызов окна свойств". Если окно свойств на экране присутствует, сделайте его активным, щелкнув в нем или нажав клавишу <F4>.
3. **Наберите новое имя.**

   Поскольку в данном случае в окне есть только одно свойство, предваряющий печатание щелчок в строке свойства (Name) становится излишним.

Краткое знакомство с окнами для отладки

Хотя подробно приемы отладки обсуждаются в главе 9, обзор окон редактора был бы неполным без хотя бы краткого знакомства с окнами, предназначенными для отладки. Таких окон в редакторе Visual Basic четыре.

- ✓ **Окно Immediate** {Окно немедленного выполнения команд) позволяет выполнять отдельные VBA-операторы прямо "на лету" (рис. 5.18). Можно также для отладки заставить программу печатать в этом окне сообщения, характеризующие процесс выполнения программы.

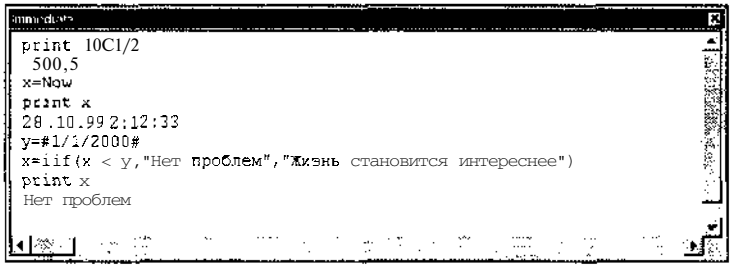


Рис. 5.18. Выполнение VBA-кода в окне немедленного выполнения команд

- ✓ **Окно Locals** (Окно локальных переменных) предлагает сведения об объектах и переменных из выполняемой в данный момент процедуры.
- ✓ **Окно Watches** (Окно контролируемых выражений) отображает текущие значения указанных вами объектов и переменных. Эти объекты и переменные могут принадлежать любой части вашей программы, а не только выполняемой в данный момент процедуре.
- ✓ **Окно Call** Stack (Окно стека вызовов) предлагает список процедур, которые выполнялись в программе вплоть до того момента, когда вы отобразили это окно.

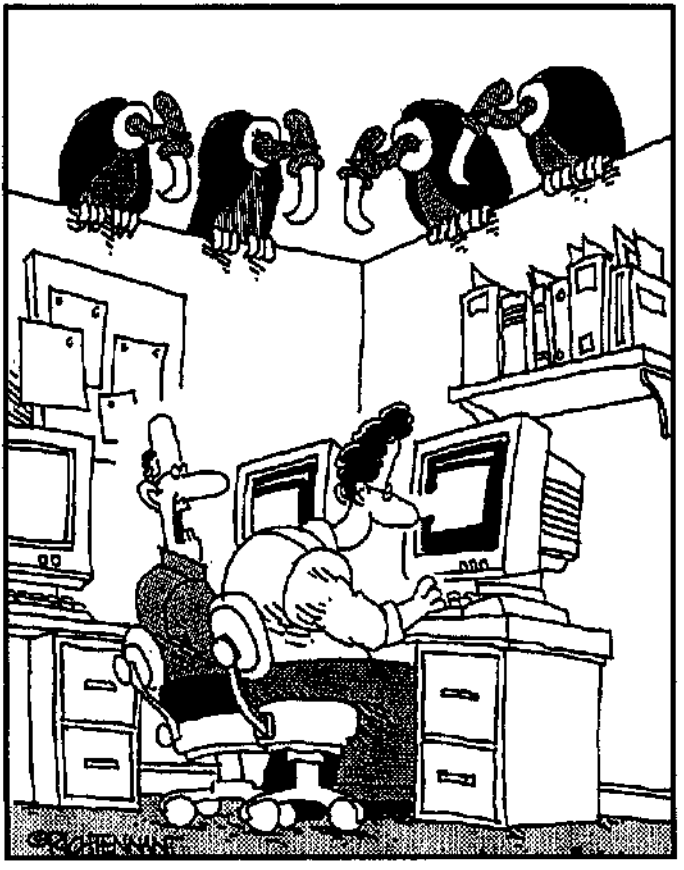
Три из этих окон — исключение составляет окно стека вызовов — можно открыть в любое время. Окно стека вызовов можно увидеть только в *режиме паузы* (break mode) при выполнении программного кода. Правда, увидеть что-либо полезное в окне локальных переменных или в окне контролируемых выражений тоже можно только в режиме паузы.

С такими отклонениями от прямого пути изложения я могу спокойно позволить себе упомянуть и о том, как вызвать на экран окна для отладки. Ясно, что всегда можно открыть меню View (Вид) и выбрать в нем соответствующий пункт. Однако для более быстрого достижения цели лучше использовать следующие варианты.

Окно	Кнопка	Комбинация клавиш
Immediate	В панели инструментов Debug	<Ctrl+G>
Locals	В панели инструментов Debug	Нет
Watches	В панели инструментов Debug	Нет
Call Stack	В панели инструментов Debug или справа вверху окна Locals	<Ctrl+L>

Часть I

Курс программирования на VBA



"У тебя никогда не возниколо предчувствия,
что этот лроект долго не протянет?"

В этой части...

Если вы хотите окунуться в море VBA-программирования, эта часть книги будет вашим пляжем. В то время как в других частях обсуждаются главные принципы и приемы программирования, здесь закладываются основы тех знаний, которые подготовят вас к штормовым будням программистской работы. Уверен, что в будущем к этой части книги вы будете обращаться за поддержкой куда чаще, чем к другим.

В главе 6 обсуждается общая структура VBA-программы в целом, подробно объясняется назначение каждого из ее строительных блоков. Здесь же говорится о правилах и соглашениях присвоения имен объектам в VBA, предлагаются приемы, с помощью которых можно сделать программный код аккуратнее и удобнее для восприятия.

В главе 7 вас ждет достаточно информативный обзор констант и переменных VBA — именованных рычагов, с помощью которых вы манипулируете информацией в своих программах.

В главе 8 изучаются приемы управления последовательностью действий, выполняемых программой, — приемы, с помощью которых вы сможете сделать свои программы немного более "интеллектуальными".

Наконец, поскольку ничто и никогда не работает так, как планируется, в главе 9 идет речь о том, как избежать появления возможных ошибок в программе, а также о способах выявления и исправления все-таки появившихся.

И завершит часть II глава, посвященная формам. В самостоятельных версиях Visual Basic формы будут единственным средством взаимодействия пользователя с программой во время ее выполнения. В VBA формы занимают не столь исключительное положение ввиду того, что для доступа к VBA-программе можно использовать и пользовательский интерфейс содержащего эту программу VBA-приложения.

Тем не менее пользовательские диалоговые окна в VBA-программах требуются тоже нередко, и в этой части мы обсудим основные средства и приемы создания форм в VBA. В главе 10 обсуждаются основы проектирования форм и работа с элементами управления.

Глава 6

Анатомия выдающейся VBA-программы

В этой главе...

- Подробности о компонентах VBA-программ и о связях между этими компонентами
- Когда следует создавать новый модуль и как в нем размешать программный код
- Как создавать подпрограммы и функции
- Обзор всех типов операторов VBA
- Правила VBA, касающиеся выбора имен для переменных, форм, модулей и других элементов программы
- Выбор имен, облегчаюших понимание программного кода
- Использование отступов в программном коде
- Разделение операторов на несколько строк
- Добавление комментариев для объяснения программного кода

В предыдущих главах, когда мне приходилось говорить о различных компонентах программного кода в VBA-программах, я использовал импровизированные и чисто описательные формулировки. Теперь пришло время дать определения этих компонентов по крайней мере на уровне, близком к формальному. Сначала мы с вами классифицируем все элементы в иерархии программного кода VBA, а затем обсудим их по отдельности более подробно.

Затем мы поговорим о том, как создать читабельный, понятный и работоспособный программный код. Здесь вы узнаете о правилах и соглашениях, касающихся выбора имен для переменных, процедур и других объектов. Здесь же вы найдете и рекомендации по форматированию программного кода для улучшения его восприятия, а также по добавлению в программный код поясняющих комментариев (чтобы вы в будущем не мучались над вопросом, почему эта странная процедура написана именно так, а не иначе).

Строительные блокипрограммы

VBA-программа — это не случайный набор инструкций компьютеру. Строки программного кода организованы в процедуры, которые размещаются в модулях, а последние, в свою очередь, размещаются в проектах.

Определение программы

Так что же такое *программа?* Говоря формально, программа — это цельная, способная функционировать (или неправильно функционировать) совокупность программного кода. Программа должна содержать один или несколько операторов, которые выполняются в порядке, заданном программистом.

Но программа не является официальным объектом VBA. VBA распознает по именам процедуры, модули и проекты, но не программы. Любая VBA-программа обязательно должна содержать хотя бы одну процедуру — просто потому, что VBA может выполнять только операторы, помещенные в процедуры, — но зато программа может управлять двумя или сразу многими процедурами, помещенными в один или несколько модулей из одного или нескольких проектов.

В этой книге в основном речь идет о программах, имеющих дело только с одним проектом. Краткий обзор использования в VBA-программе нескольких проектов вы найдете в главе 14.

Пример программы

Чтобы сделать обсуждение иерархии VBA-элементов немного менее абстрактным, давайте рассмотрим приведенный ниже программный код модуля. Этот модуль содержит все упомянутые выше элементы (за исключением проекта, поскольку модули содержатся в проектах, а не наоборот). После текста модуля идет обсуждение составляющих его элементов.

Думаю, сначала нужно объяснить, что делают в этом примере программы. Сначала создается совокупность (*массив*) из шести целых величин, которым приписываются случайные значения от 1 до 1000. Затем после подсчета числа значений, превышающих определенное пороговое значение, а именно значение 500, полученный результат отображается в небольшом диалоговом окне.


```
Option Explicit
Const Maximum As Integer = 500
Const HowMany As Integer = 5
Dim ListOfNumbers() As Integer

Sub MAIN()

    Dim ItemsInList, n, x As Integer
    ReDim ListOfNumbers(HowMany)

    Randomize

    For x = 0 To HowMany
        ListOfNumbers(x) = Int((1000 * Rnd) + 1)
    Next x

    n = CountBigNumbers()
    MsgBox ("There were " & n & " values greater than " & _
     Maximum)

End Sub

Function CountBigNumbers()

    Dim Counter As Integer, y As Integer

    Counter = 0

    For y = 0 To HowMany
        If ListOfNumbers(y) > Maximum Then
            Counter = Counter + 1
        End If
```

```
    Next y

    CountBigNumbers = Counter

End Function
```

Весь этот блок профаммного кода в своей совокупности представляет собой *модуль.* Он состоит из ряда операторов, организованных в три раздела. Это раздел объявлений, начинающийся оператором `Option Explicit`, процедура типа Sub, начинающаяся оператором Sub MAIN (). и процедура типа `Function`, начинающаяся оператором `Function CountBigNumbers ()`. Большинство операторов этого модуля занимают по одной строке, но оператор в строке, начинающийся с выражения `MsgBox`, заканчивается в следующей строке.

Иерархия VBA

Теперь, после знакомства с приведенным выше примером программы, вам легче будет понять следующие определения и описания строительных блоков программного кода VBA.

- ✓ *Оператор* — это наименьшая, способная выполняться единица VBA-кода. Оператор может объявлять или определять переменную, устанавливать параметр компилятора VBA или выполнять какое-либо действие в профамме. У допустимого оператора много сходства с законченным предложением — оператор должен содержать правильный набор "частей речи", иначе это не оператор вообще.
- ✓ *Процедура* — это наименьшая единица программного кода, на которую можно ссылаться по имени. Это также наименьшая единица программного кода, которая может выполняться независимо. VBA распознает два главных типа процедур —`Sub` и `Function`. Любая процедура содержит один или несколько операторов, помещенных между двумя специальными операторами,— объявлением процедуры в начале и оператором завершения процедуры (`End Sub` или `End Function`) в конце.
- ✓ *Модуль* — это именованная единица, состоящая из одной или нескольких процедур, а также объявлений, относящихся ко всем процедурам в модуле. Хотя VBA и допускает размещение всех процедур в одном модуле, имеет смысл разместить процедуры в нескольких модулях в соответствии с выполняемыми этими процедурами задачами, чтобы с ними было проще работать.
- ✓ В VBA два типа модулей. Чаще всего используется *стандартный модуль,* который содержит программный код, предназначенный непосредственно для выполнения. Другим типом модуля является *модуль класса,* в котором определяются пользовательские объекты с их свойствами и методами. Краткое описание модулей класса вы найдете в главе 14.

- ✓ *Проект* состоит изо всех модулей, форм и связанных с приложением объектов, относящихся к некоторому документу, вместе с самим документом.

Откуда берутся проекты, мама?

Их приносит аист. Ну, ладно, это определенно не так, но главное в том, что для создания проекта не требуется выполнять никаких специальных действий. Каждый документ VBA-приложения автоматически является проектом. Конечно, проект, состоящий из одного документа, не содержит ни профаммного кода, ни форм до тех пор, пока вы не создадите их в редакторе Visual Basic или не запишете макрос в приложении. (Приемы работы с проектами обсуждаются в главе 5.)

Все о модулях

В иерархии элементов программного кода VBA модули стоят на ступеньку ниже проектов. Модуль хранит одну или несколько процедур, а также раздел объявлений, состоящий из операторов, применимых ко всему модулю.

Планирование модулей

Нельзя сказать, что проблема организации модулей слишком сложна. Имеет смысл подумать только над тем, сколько модулей следует создать и какие процедуры должны в них войти. Вот те моменты, о которых нужно при этом помнить.

- ✓ Процедуры могут *вызывать* или выполнять процедуры, хранящиеся в других модулях. Один модуль также может использовать переменные, объявленные в другом модуле.
- ✓ Использование процедур и переменных из других модулей немного усложняет задачу программирования. Например, можно вызвать процедуру из другого модуля, просто напечатав ее имя. Однако, если имя той процедуры совпадает с именем процедуры из вызывающего модуля, необходимо напечатать перед именем процедуры имя содержащего ее модуля, например `ДругойМодуль.НужнаяПроцедура`. (Я вообще рекомендовал бы использовать формат `Модуль.Процедура` всегда, даже если никаких совпадений имен нет.)

В общем, следует размещать связанные каким-либо образом процедуры в одном модуле. Как правило, один модуль содержит все процедуры одной целой VBA-программы. Это упрощает программирование, поскольку не приходится иметь дело с вышеописанными сложностями с вызовом процедур и переменных из других модулей.

И все же, если вы планируете создание процедур, которые будут использоваться более чем одной программой, имеет смысл организовать эти процедуры в модули по типу выполняемых ими задач, например `МатематическиеПроцедуры` ИЛИ `БлокОбработкиТекста`. Можно создавать и модули типа `НовыеПроцедуры` или `СтарыеПроцедурыНаВсякийСлучай`. Подробности вы найдете в разделе "Выбор имен" дальше в настоящей главе.

Добавление нового модуля в VBA-проект

Чтобы создать новый модуль в редакторе Visual Basic, сначала убедитесь, что вы работаете с нужным проектом. В окне проводника проектов выделите либо сам проект, либо один из его компонентов — не имеет значения, какой (это может быть, например, форма, объект приложения или модуль). После этого можно добавить модуль в проект любым из следующих способов.

- ✓ Щелкните на кнопке Insert Module (Добавить модуль) (ее изображение вы видите рядом). Это многофункциональная кнопка, предназначенная для добавления в проект различных объектов. Если картинка добавления модуля не видна, а вместо нее видна, например, картинка добавления формы, щелкните на узкой полоске со стрелкой около главной части кнопки и выберите нужный пункт из раскрывшегося меню.
- ✓ Щелкните правой кнопкой мыши в окне проводника проектов — не забудьте при этом проследить, чтобы курсор находился в рамках нужного проекта, — и выберите Insert Module (Добавить модуль) из появившегося контекстного меню.
- ✓ Выберите Insert⇨Module из меню.

В результате редактор Visual Basic автоматически откроет окно программного кода нового модуля. Созданному модулю будет также назначено типичное имя. Чтобы изменить это имя, напечатайте новое имя в окне свойств (подробности в главе 5).

Что к чему в новом модуле

В окне программного кода для нового модуля есть только один раздел — Declarations (Объявления). Чтобы выяснить, в каком из разделов вы находитесь, взгляните на текст в окне списка справа вверху окна программного кода.

Два типа операторов можно поместить в раздел объявлений модуля.

- ✓ Объявления переменных, констант и пользовательских типов данных. Такие объявления сообщают компилятору имя и тип каждого из объектов (но не их значения). Переменные и константы, объявленные в разделе объявлений модуля, могут использоваться в любой процедуре этого модуля.
- ✓ Параметры компилятора, с помощью которых можно управлять работой компилятора VBA.

Операторы присваивания или выполняемые операторы нельзя размещать в разделе объявлений. Например, в разделе объявлений нельзя задать *значение* переменной, поскольку для этого требуется оператор присваивания, который можно разместить только внутри процедуры где-нибудь в другом месте модуля. (Различия между выполняемыми операторами, операторами присваивания, объявлениями и параметрами компиляции будут обсуждаться ниже, в разделе "Использование операторов".)

Операторы из раздела объявлений имеют и другое, довольно забавное, название — *программный код уровня модуля.*

Каждая добавляемая в модуль процедура рассматривается как новый раздел. После того как процедура добавлена, ее имя появляется в списке, окно которого находится вверху справа в окне программного кода. Тем самым вы получаете возможность сразу перейти к процедуре, выбрав имя этой процедуры из списка.

Стандартные модули и модули с классом

Большинство создаваемых VBA-модулей — это *стандартные модули.* В предыдущих версиях VBA и в Visual Basic стандартные модули назывались *модулями программного кода.* Такие модули содержат объявления переменных и констант, определения пользовательских типов данных и установки параметров компиляции, а также выполняемые операторы, которые и делают практическую работу. В стандартном модуле можно использовать любой объект, к которому можно получить доступ, но нельзя создать новый тип объекта.

Модули класса, напротив, предназначены для того, чтобы определять в них пользовательские объекты. Создав модуль класса для своего собственного объекта, вы можете затем наполнить его программным кодом, задающим свойства и методы этого объекта. После этого в программном коде из других модулей можно использовать свойства и методы нового объекта точно так же, как это делается со встроенными объектами VBA. Здесь у меня нет возможности углубляться в детали, но хочу заметить, что создание своих собственных объектов служит очень мощным средством программирования.

Создание процедур

Процедуры — это критические функциональные единицы VBA-программы в том смысле, что вы можете выполнить только программный код, содержащийся в какой-либо процедуре.

Основными VBA-процедурами являются процедуры двух типов — Sub (подпрограммы) и Function (функции). Дополнительная информация о процедурах приводится ниже, во врезке "Типы процедур".

Типы процедур

Почти весь создаваемый вами программный код будет содержаться в процедурах всего двух типов - Sub и Function. Процедуры обработки событий, выполняемые в VBA, когда происходит какое-либо событие, например щелчок кнопки мыши, являются специальными процедурами типа **Sub.** В VBA есть еще один тип процедур – это процедуры типа Property (процедуры свойств).

Вот краткое описание особенностей процедур различных типов.

- ✓*Процедура типа* Sub (подпрограмма) - универсальная процедура для выполнения различных заданий в VBA. Только процедуры типа Sub можно выполнять независимо с помощью приемов, обсуждавшихся в главе 4. Кроме того, одна процедура типа Sub может *вызывать* (т.е. выполнять) другую.
- ✓ *Процедура типа* Function (функция) тоже может выполнять любой VBA-оператор. Отличие от процедур типа Sub состоит в том, что в данном случае вычисляется некоторое значение, которое *возвращается* в ту процедуру, откуда процедура типа Function была вызвана.
- ✓*Процедура обработки события* (event procedure) - это процедура типа Sub специального назначения. Признаю, что об этом я уже говорил. Но зато я не говорил о том, что подробности о процедурах обработки событий вы найдете в главе 14.
- ✓ *Процедура типа* Property (процедура свойства) выясняет или устанавливает значение свойства пользовательского объекта. О процедурах свойств не будет идти речь аж до главы 14, но здесь они упоминаются просто потому, что вы непременно увидите их среди опций диалогового окна Add Procedure (Добавление процедуры).

Строго говоря, макрос в VBA- это процедура типа Sub, не имеющая (т.е. не требующая) параметров. Макросы образуют единственный класс процедур типа Sub, способных выполняться непосредственно путем вызова по имени либо из редактора Visual Basic, либо из VBA-приложения. Чтобы выполнить процедуру типа Sub с параметрами, ее необходимо вызвать из другой процедуры. Работа с аргументами будет обсуждаться в этой же главе в разделе "Привлекательные аргументы".

Каркас процедуры

Вот два примера процедур, по одной каждого из типов (Sub и Function):

```
Public Sub СубМарина()
    MsgBox "Поднять перископ!"
End Sub

Public Function ФункШин(ДатаРождения As Date)
    ФункШин = DateDiff("yyyy", ДатаРождения, Date)
End Function
```

Как видите, каждая из процедур имеет начинающий процедуру оператор ее объявления, как минимум одну строку программного кода и завершающий оператор End. Подробнее эти элементы будут рассмотрены чуть позже в разделах "Процедуры типа Sub" и "Процедуры типа Function".

Создание новой процедуры

Прежде чем создавать новую процедуру, нужно открыть окно программного кода для того модуля, в котором вы собираетесь эту процедуру разместить. Создайте новый модуль в соответствии с инструкциями, приведенными выше в разделе "Добавление нового модуля в VBA-проект", или откройте любой уже существующий модуль, дважды щелкнув на строке с его именем в окне проводника проекта.

После того как окно программного кода модуля будет открыто и станет активным, можно приступить к добавлению новой процедуры. Для начала просто вставьте в модуль оператор объявления процедуры и оператор ее завершения, т.е. оператор End. Это можно сделать двумя способами.

- ✓ Выбрать Insert⇨Procedure и заполнить появившееся при этом диалоговое окно.
- ✓ Ввести соответствующие операторы вручную.

Печатать или использовать диалоговое окно?

Я рекомендую печатать каркас процедуры вручную просто потому, что использование диалогового окна не дает никакого выигрыша во времени, разве что вы печатаете *исключительно* медленно. Кроме того, подход "сделайте сами" позволяет выбрать место размещения процедуры в модуле, которую все равно не удастся заполнить с помощью диалогового окна. Нужно только щелкнуть в том месте, где должна появиться процедура, напечатать строку с ее объявлением и нажать <Enter>. Вводить завершающие операторы End Sub и End Function не потребуется — редактор Visual Basic подставит подходящий оператор за вас.

Правда, возможно, что в самом начале освоения VBA вы будете чувствовать себя увереннее с диалоговым окном. Чтобы использовать этот подход, щелкните на кнопке Insert Procedure (Добавить процедуру). (Опять же, наверное, вам придется сначала щелкнуть на этой небольшой полоске со стрелкой рядом с кнопкой Insert и только затем выбрать Procedure из раскрывшегося меню.) Можно также просто выбрать Insert⇨Procedure из меню.

Появится диалоговое окно Add Procedure (Добавление процедуры). В этом окне нужно напечатать имя процедуры в поле Name (Имя), затем выбрать с помощью переключателей Type (Тип) и Scope (Область видимости) соответственно тип и область видимости, а также установить, если нужно, флажок All Local variables as Statics (Считать все локальные переменные статическими). Я знаю, что до сих пор ничего не говорил об *области видимости* и о *статических переменных,* но на все эти вопросы вы найдете ответы ниже, в разделе "Обзор области видимости".

После щелчка на кнопке OK редактор Visual Basic вставит пару из объявления процедуры и оператора End в конец модуля, не обращая внимания на то, где находился текстовый курсор до того, как вы открыли диалоговое окно.

Наполнение каркаса процедуры

Создание операторов объявления и завершения процедуры, несомненно, является самым простым делом в процессе создания новой процедуры. А главная работа состоит в печатании операторов VBA, которые должны находиться между открывающим и закрывающим операторами. В этом разделе не предполагается вдаваться в детали того, что именно следует там печатать. Достаточно сказать, что операторы внутри процедуры будут выполняться в порядке их появления до тех пор, пока не встретится оператор, предписывающий VBA перейти в другое место.

Процедуры типа Sub

Термин "процедура типа Sub" кажется мне довольно неуклюжим, но он достаточно точно отражает суть дела. В некоторых языках программирования субпроцедурами, или подпрограммами, называются процедуры, вызываемые главными процедурами. В VBA, хотя проце-

дуры типа Sub и могут вызываться другими процедурами, главная процедура программы *всегда* является процедурой типа Sub. Это судьба.

Но хватит моего ворчания — перейдем к фактам. Вот пример процедуры типа Sub с объявлением в начале, завершающим оператором — • в конце и несколькими операторами между ними:

```
Public Sub ПриятнаяПроцедура()
     Dim ДоброеСообщение As String
     ДобрсеССобщение = "Добрый день!"
     MsgBox ДоброеСообщение
      (другие операторы)
...
End Sub
```

Первая строка, объявление процедуры типа Sub, решает две жизненно важные задачи. Во-первых, она говорит, где начинается процедура, так что VBA теперь знает, откуда начинать выполнение программного кода при вызове процедуры. Во-вторых, объявляет характеристики процедуры, обсуждение которых предполагается в следующем разделе.

Каждая процедура типа Sub должна заканчиваться оператором End Sub, который дает VBA знать, где следует прекратить выполнение программного кода.

Элементыобъявленияпроцедурытипа Sub

В операторе объявления процедуры первый термин Public определяет область видимости процедуры. Для области видимости можно указать либо Public, либо Private. При этом Public подразумевается по умолчанию, так что это ключевое слово можно пропустить. Логично, что пропуск ключевого слова Public делает область видимости такой, которой вообще не нужно интересоваться. Снова повторю, что подробно область видимости описана в разделе "Обзор области видимости".

Далее идет ключевое слово Sub, которое просто указывает на то, что здесь определяется процедура типа Sub. За ним следует имя процедуры, которое может быть каким угодно, подчиняясь лишь правилам присвоения имен, приведенным дальше в настоящей главе.

Пропущенныеаргументы

Завершается объявление парой скобок. Зачем они нужны, если внутри них ничего нет? Эти скобки могли бы содержать *аргументы*, если бы у процедуры они были. Аргументы — это элементы данных, которые процедура намеревается получить при вызове. В данном случае процедура не имеет аргументов, поэтому она не только внешне такая приятная, но и в скобках ничего не содержит. Дальнейшее обсуждение аргументов продолжится позже в разделе "Привлекательные аргументы'".

Снова повторю, что макрос в VBA по определению представляет собой процедуру типа Sub, не имеющую аргументов. Чтобы выполнить процедуру типа Sub, у которой аргументы есть, ее нужно вызвать из другой процедуры.

Вызовпроцедуртипа Sub

Любую процедуру — независимо от того, имеет она аргументы или нет — можно выполнить, или *вызвать*, из другой процедуры. Чтобы вызвать процедуру типа Sub, используйте оператор, представляющий собой имя вызываемой процедуры, как в следующем фрагменте программного кода. Строка, в которой напечатано МойкаМоейСтаройМашины, вызывает процедуру типа Sub:

```
...
МояСтараяМашина = "Доблестная"
МойкаМоейСтаройМашины
ЧислоПосещенийМойки = ЧислоПосещенийМойки +1
...
```

Процедуры типа Function

Процедура типа `Function`, в принципе, работает так же, как и процедура типа Sub, но в данном случае ее главная задача — вычисление некоторого значения. Когда процедура типа `Function` завершит свою работу, она возвратит это значение в вызывающую процедуру, которая сможет использовать его в дальнейших вычислениях.

Вот пример процедуры типа `Function`:

```
Public Function ДеФункция(x As Integer, y As Integer)
    Dim Z AS Integer
    z = x + y
    Де Функция = x ^ z
End Function
```

Очевидно, строение процедуры типа `Function` очень похоже на строение процедуры типа Sub. Объявление начинается с необязательного ключевою слова, определяющего область видимости процедуры (в данном случае это `Public`). Далее идет ключевое слово `Function`, определяющее тип процедуры, за ним размещается имя процедуры и, наконец, ее аргументы. Аргументы будут обсуждаться в разделе "Привлекательные аргументы", но уже на этом примере вы видите, как они используются. Подобно процедуре типа Sub, в конце процедуры типа `Function` находится завершающий оператор `End Function`.

Здесь я вынужден сделать паузу, чтобы обратить ваше внимание на то, что VBA содержит немалое количество встроенных функций. Подобно процедурам типа `Function`, функции VBA возвращают в вызывающую процедуру некоторое значение. Вызываются функции и процедуры типа `Function` тоже одинаково. Единственным различием между функциями и процедурами типа `Function` является то, что для функций вам не нужно печатать программный код! Обзор функций, доступных в VBA, вы найдете в главе 11.

Отличия процедур типа Function от процедур типа Sub

Между процедурами типа `Function` и типа Sub есть одно существенное отличие: в процедуре типа `Function` обязательно где-то должен присутствовать по крайней мере один оператор, задающий значение этой функции. При этом используется имя процедуры, как будто это обычная переменная. В предыдущем примере присвоение значения осуществляется в строке `ДеФункция = x ^ z`. После того как программный код в этой строке будет выполнен, `ДеФункция` будет содержать значение, которое возвратится в вызывающую процедуру для дальнейшего использования там.

Вызов процедур типа Function

Процедуру типа `Function` можно выполнить, только вызвав ее из другой процедуры. Как правило, это делается путем присваивания имени данной функции некоторой переменной. В следующем примере переменная `ZCтепень` получает значение, возвращенное процедурой `ДеФункция`:

```
...
ZСтепень = ДеФункция(3, 4)
...
```

После выполнения этого оператора переменная ZСтепень будет содержать значение, вычисленное процедурой ДеФункция, — процедурой типа Function — и этим значением будет 3^7. Обратите внимание, что в данном случае процедуре передаются аргументы, заключенные в скобки после имени процедуры.

Привлекательные аргументы

Аргументы представляют значения, которые предполагается *передавать* от одной процедуры к другой. Аргументы назначаются процедуре тогда, когда необходимо, чтобы эта процедура изменяла свое поведение в зависимости от тех значений, которые она получает при ее вызове. Аргументы могут иметь и процедуры типа Sub, и процедуры типа Function.

Рассмотрим снова процедуру ДеФункция:

```
Public Function ДеФункция(x As Integer, y As Integer)
    Dim z As Integer
    z = x + y
    ДеФункция = x ^ z
End Function
```

У этой процедуры два аргумента— x и y. Как видно из программного кода, аргументы процедуры являются частью ее объявления. Аргументы размещаются в скобках сразу после имени процедуры в *списке аргументов,* где для каждого из аргументов задается имя и тип данных (подробности — ниже, в разделе "Создание процедур с аргументами").

Если у процедуры есть аргументы, то она будет требовать эти аргументы при выполнении своей работы. Внутри такой процедуры аргументы играют практически ту же роль, что и переменные, объявляемые самым обычным образом.

Посмотрите внимательнее на программный код процедуры ДеФункция. После объявления переменной z процедура вычисляет значение этой переменной, равное сумме двух аргументов x и y. В следующей строке, чтобы вычислить возвращаемое значение самой процедуры, аргумент x возводится в степень z. Как видите, к, y и z играют одинаковые роли. Если хотите, можно возвести z в степень x.

Полезныеаргументы

Но если аргументы так похожи на обычные переменные, зачем же вообще их использовать? В действительности их можно и не использовать — все, что делают аргументы, можно сделать с помощью обычных переменных. Но аргументы упрощают использование процедур и восприятие программного кода.

Пришло время рассмотреть пару примеров для сравнения. Сначала рассмотрим процедуры, использующие аргументы Модель и ГодВыпуска для передачи значений:

```
Public Sub ВызывающаяПроцедура()
    ДоходОтСтаройМашины = ПродажаСтаройМашины("Rambler _
    Classic",1962)
End Sub

Public Function ПродажаСтаройМашины(Модель As String, _
          ГодВыпуска As Integer)
Dim ВозрастнойФактор As Integer
Dim ЦеновойФактор As Single
```

```
    ВозрастнойФактор = ГодВыпуска - 1900
    If Модель = "Rambler Classic" Then
        ЦеновойФактор = .001
    Else If Модель = "Dodge Dart" Then
        ЦеновойФактор = .005
    Else If ...
        (другие операторы)
    End If
    ПродажаСтаройМашины = ВозрастнойФактор * _
    ЦеновойФактор * 1000
End Function
```

Здесь процедура с именем ВызывающаяПроцедура использует процедуру типа Function с именем ПродажаСтаройМашины для вычисления результата с именем ДоходОтСтаройМашины. Раз процедура типа Function использует аргументы, вы можете сообщить ей непосредственно, какую машину вы продаете и какого она года выпуска.

Теперь рассмотрим две процедуры, выполняющие ту же работу без аргументов. Заметим, что в данном случае необходимо объявить две переменные на уровне модуля, вне процедур, чтобы этими переменными могли пользоваться обе процедуры.

```
Dim Модель As String
Dim ГодЗыпуска As Integer
Public Sub ВызывающаяПроцедура2()
    Модель = "Dodge Dart"
    ГодВыпуска = 1963
    ДоходОтСтаройМашины = ПродажаСтаройМашины2()
End Sub
Public Function ПродажаСтаройМашины2()
Dim ВозрастнойФактор As Integer
Dim ЦеновойФактор As Single
    ВозрастнсйФактор = ГодВыпуска - 1900
    If Модель = "Rambler Classic" Then
        ЦеновойФактор = .001
    Else If Модель = "Dodge Dart" Then
        ЦеновойФактор = .005
    Else If ...
        (другие операторы)
    End If
    ПродажаСтаройМашины = ВозрастнойФактор * _
    ЦеновойФактор * 1000
End Function
```

Такзачемжевозражать?

Пока эти примеры не улетучились из вашей головы, я приведу список преимуществ, которые дает использование аргументов для обмена информацией между процедурами.

- ✓ При чтении программного кода процедуры именно аргументы ясно указывают, какие значения требуются процедуре от других частей программы для выполнения работы.
- ✓ При создании программного кода аргументы помогают уменьшить число переменных, которые требуется создать (вне всех процедур в разделе объявлений модуля). Одной очевидной проблемой переменных уровня модуля является то, что при создании использующей их процедуры не видно определений таких переменных.
- ✓ При вызове процедуры с аргументами VBA *заставляет* вас определить значения для таких аргументов. Это гарантирует, что процедура получит именно те значения, которые ей требуются.

Создание процедур с аргументами

Чтобы создать процедуру с аргументами, поместите аргументы в скобки после имени процедуры в ее объявлении. Например:

```
Романист(Заглавие As String, Страницы As Integer, _
         СрокЗавершения As Date)
```

Отсюда видно, что тип каждого из аргументов задается следующим за именем аргумента сочетанием As тип, где тип может быть любым допустимым в VBA типом данных или классом объекта. Если указать только имя аргумента и не указать тип, то VBA припишет такому аргументу тип Variant. (Да знаю я, знаю, что еще не обсуждал типы данных и классы объектов, но вы можете найти все о них в главах 7 и 12 соответственно.)

Вызов процедур с аргументами

Чтобы где-нибудь в программном коде вызвать процедуру с аргументами, просто сразу за именем этой процедуры напечатайте значения каждого из аргументов. Значения должны быть перечислены в том же порядке, в котором соответствующие аргументы заданы в объявлении процедуры.

Единственной возможной сложностью при этом могут стать правила использования или неиспользования скобок. Правила эти следующие.

- ✓ Скобки для аргументов *обязательны* при вызове процедуры типа Function для получения возвращаемого значения.
- ✓ Скобки *можно опустить* в случае вызова процедуры типа Sub, а также в случае, когда в программном коде, вызывающем процедуру типа Function, возвращаемое значение не используется,

Организация процедур

Любую VBA-программу, независимо от ее длины, можно оформить в виде одной-единственной процедуры. Так зачем же создавать себе лишние заботы, разбивая программу на процедуры? Главная причина в том, что это существенно упрощает контроль над выполняемой работой по программированию.

С ростом объема программы еще быстрее растет вероятность того, что вы что-нибудь в ней упустите. Если программа разделена на процедуры, имена которых соответствуют выполняемым процедурами задачам, вы получаете возможность сконцентрироваться на той части программы, которая нужна вам в данный момент. Не забывайте, что в окне программного кода справа вверху имеется список, из которого можно выбрать имя процедуры и сразу перейти к этой процедуре, где бы внутри модуля она ни находилась.

Еще одним подтверждением пользы оформления блоков программного кода в виде процедур служит возможность при этом устранить повторный ввод одних и тех же блоков. Например, если в разных местах программы используется одна и та же последовательность операторов, то можно упростить программу и уменьшить ее размеры, поместив повторно используемые операторы в процедуру. Такую процедуру можно будет вызывать по имени из любой части программы.

Кроме того, создание процедуры для повторно используемой последовательности операторов сокращает объем довольно нудной работы по печатанию программного кода, а также позволяет избежать опечаток, которые вы можете допустить при вводе (точнее, *обязательно* допустите, если будете печатать фрагменты одинакового программного кода повторно). Если повторяющиеся операторы представить в виде процедуры, то и корректировать придется только одну эту процедуру.

Обзор области видимости

Каждая VBA-процедура имеет свою вполне определенную *область видимости*. Область видимости определяет, из какой части программы вы можете вызвать эту процедуру, а из какой — нет. Можно сказать, что область видимости отвечает за то, какая часть программы сможет "увидеть" данную процедуру.

Область видимости процедуры может быть одной из следующих трех типов.

- ✓ По умолчанию процедуры VBA (за исключением процедур обработки событий) рассматриваются как *открытые* (public). Это значит, что вы можете вызывать их из любой части программы — из того же модуля, из другого модуля и даже из другого проекта (конечно, если программа использует несколько проектов, что обсуждается в главе 14).
- ✓ Если нужно, можно объявить процедуру *локальной* (private). Локальная процедура будет видима (если вы пожелаете) только внутри содержащего ее модуля. Другими словами, вы сможете вызвать локальную процедуру из других процедур того же модуля, но не из процедур, размещенных в других модулях.
- ✓ В VBA-программах, использующих несколько проектов, можно создавать процедуры, доступные изо всех модулей в рамках данного проекта, но не из других проектов.

Точно такие же области видимости задаются и переменным (подробно об этом — в разделе "Задание области видимости переменной" главы 7).

Задание области видимости процедуры

Чтобы задать область видимости процедуры, нужно в начале объявления процедуры просто напечатать ключевое слово Public или Private. Взгляните на следующие примеры:

```
Public Sub IKneadYou()
...(операторы процедуры)
End Sub

Private Function IKneedYou()
.. . (операторы процедуры)
End Function
```

Поскольку по умолчанию процедуры предполагаются открытыми, ключевое слово Public указывать для этого не обязательно. Однако, если в программе есть хотя бы одна локальная процедура, я рекомендовал бы явно определять и открытые процедуры, чтобы с первого взгляда определить область видимости любой из процедур.

Чтобы ограничить область видимости открытой процедуры рамками одного проекта и сделать ее недоступной для других проектов, поместите в раздел объявлений соответствующего модуля оператор Option Private Module. Подробности об использовании таких операторов вы найдете ниже, в разделах "Объявления" и "Параметры компилятора".

Использование локальных процедур

Объявление процедуры локальной помогает избежать многих ошибок. Поскольку в таком случае можно вызывать процедуру только в рамках содержащего ее модуля, легче контролировать условия, которые складываются во время вызова (эти условия включают значения переменных, используемых этой процедурой).

Объявить процедуру локальной несложно, но зачем беспокоиться? В конце концов, VBA же *не требует* вызывать процедуру только потому, что она открыта!

Главной причиной является повышение надежности. Вы можете забыть, что данная процедура предназначена для использования только внутри модуля. Если процедура локальна, VBA не позволит вам вызвать ее из другого модуля, когда вы попытаетесь это сделать. А в программах значительных объемов и сложности уменьшение области доступа к процедуре еше и помогает контролировать организацию программы.

использование операторов

Процедуры состоят из операторов — наименьших жизнеспособных единиц программного кода. Как правило, операторы занимают по одной строке программного кода, и в каждой строке обычно содержится только один оператор, но, как будет показано немного позже, это не обязательно. В VBA четыре типа операторов: *объявления, операторы присваивания*, *выполняемые операторы* и *параметры компилятора*.

Объявления

Объявление — это оператор, сообщающий компилятору VBA о ваших намерениях по поводу использования в программе именованного объекта (переменной, константы, пользовательского типа данных или процедуры). Кроме того, объявление задает тип объекта и обеспечивает компилятору дополнительную информацию о том, как использовать данный объект. Объявив объект, вы можете использовать его в любом месте программы.

Рассмотрим пример объявления *переменной*. Переменные — это именованные значения, которые могут изменяться во время выполнения программы (подробно об этом — в главе 7). В следующем операторе объявляется переменная с именем МоеЛюбимоеЧисло и заявляется о том, что значение, которое она будет содержать, должно быть целым:

```
Dim МоеЛюбимоеЧисло As Integer
```

В главе 7 обсуждаются также *константы,* представляющие собой именованные значения, которые не меняются. Следующий оператор создает *строковую* константу (текст) с именем НеизменныйТекст, представляющую собой набор символов Вечность:

```
Constant НеизменныйТекст = "Вечность"
```

Объявление переменной или константы можно поместить либо в разделе объявлений модуля, либо внутри конкретной процедуры. Выбор места зависит от того, какой должна быть область видимости. Здесь же вы найдете несколько примеров объявления переменных.

Следующим оператором объявляется пользовательский тип данных с именем Самоделкин, определяя его как структуру, включающую строковую переменную с именем Имя и переменную типа Date с именем ДеньРождения. В данном случае объявление займет несколько строк:

```
Type Самоделкин
     Имя As String
     ДеньРождения As Date
End Type
```

Следующее объявление создает процедуру типа Sub с именем СкрытаяПроцедура, говоря о том, что эта процедура будет локальной в смысле области видимости. Завершающий процедуру оператор End Sub считается частью объявления. Предполагается, что операторы, составляющие тело процедуры, вы должны напечатать сами:

```
Private Sub СкрытаяПроцедура()
     (здесь должны размещаться операторы процедуры)
End Sub
```

Еще несколько примеров объявлений процедур вы уже встречали в разделе "Создание процедур".

Операторы присваивания

Операторы присваивании приписывают переменным или свойствам объектов конкретные значения. Такие операторы всегда состоят из трех частей: имени переменной, или свойства, знака равенства и *выражения,* задающего нужное значение.

Термин *выражение* я определю чуть позже, но сначала приведу несколько примеров операторов присваивания. Следующий оператор присваивает переменной МоеЛюбимоеЧисло значение суммы переменной ДругоеЧисло и числа 12:

```
МоеЛюбимоеЧисло = ДругоеЧисло + 12
```

Вот оператор, устанавливающий свойство Color (Цвет) объекта AGraphicShape равным Blue (Синий) в предположении, что Blue является именованной константой, представляющей числовой код соответствующего цвета:

```
AGraphicShape.Color = Blue
```

А в следующем операторе, чтобы задать значение переменной КвадратныйКорень, для текущего значения переменной МоеЛюбимоеЧисло вызывается функция Sqr — встроенная функция VBA для вычисления квадратного корня:

```
КвадратныйКорень = Sqr(МоеЛюбимоеЧисло)
```

Перед тем как привести примеры, я использовал термин *выражение.* В VBA выражением называется любой фрагмент программного кода, задающий некоторое числовое значение, строку текста или объект. Выражение может содержать любую комбинацию чисел или символов, констант, переменных, свойств объектов, встроенных функций и процедур типа Function, связанных между собой *знаками операций* (например, + или *). Вот несколько примеров выражений.

Выражение	Значение
3,14	3,14
AGraphicShape	5 (в предположении, что объект AGraphicShape представляет пятиугольник)
(12 - Sqr(x))/5	2 (в предположении, что x=4)
"Розы красные, " & "фиалки фиолетовые"	Розы красные, фиалки фиолетовые

Выполняемые операторы

Выполняемые операторы делают главную работу в программе и используются для выполнения следующих задач.

- ✓ Вызов процедуры,
- ✓ Активизация метода некоторого объекта.

- ✓ Управление порядком, в котором должны выполняться другие операторы, с помощью организации циклов или выбора участка программного кода (из нескольких вариантов) для последующего выполнения.
- ✓ Выполнение одного из встроенных операторов VBA или функции.

Посмотрите внимательно на следующие примеры. Вот оператор, вызывающий для выполнения метод Rotate объекта AGraphicShape:

```
AGraphicShape.Rotate(90)
```

Следующий небольшой фрагмент программного кода содержит два выполняемых оператора. Оператор If...Then выясняет, будет ли значение переменной ПорогСлышимости больше, чем 3, и если оно больше, то программе предписано выполнить следующий выполняемый оператор, а именно тот, в котором вызывается процедура ГенераторГромкогоЗвука:

```
If ПорогСлышимости > 3 Then
    ГенераторГромкогоЗвука
End If
```

Параметры компилятора

Последний класс операторов представляет собой инструкции для управления поведением компилятора VBA. К операторам, задающим параметры компилятора, относятся следующие.

Оператор	Выполняемое действие
`Option Base число`	Установка правила нумерации массивов переменных - начинать нумерацию по умолчанию с 0 или 1 (подробности в главе 13)
`Option Compare метод`	Выбор метода, используемого VBA для сравнения строковых переменных (текста). Вместо слова "метод" можно указать Binary для сравнения на основе числового кода символов, Text - для сравнения, при котором не учитывается регистр символов, или Database (только в Access) - для сравнения в порядке, отвечающем порядку сортировки соответствующей базы данных
`Option Private Module`	В результате помещения такого оператора в раздел Declarations модуля другие *проекты* не смогут получить доступ к процедурам, переменным и константам этого модуля, даже если эти объекты объявлены как открытые (подробно об области видимости см. выше в разделе "Обзор области видимости")
`Option Explicit`	Это единственный из операторов, задающих параметры компилятора, о котором следует знать и который стоит использовать. В результате помещения этого оператора в модуль VBA запрещает использовать переменные без их предварительного явного объявления (подробности - в главе 7)

Выбор имен

В определенных рамках вы имеете возможность совершенно произвольно выбирать имена для переменных, процедур и всего другого, что вы создаете. Следующие правила применимы ко всем именованным элементам в VBA-программе. включая переменные, константы, типы данных, процедуры, модули, формы и проекты.

- ✓ Имена должны начинаться с буквы, но не с цифры. После первой буквы уже могут идти и цифры, и символ подчеркивания, как, например, в имени
- ✓ `Скрытая_Переменная3`
- V Кроме символа подчеркивания, все остальные знаки пунктуации для использования в именах в VBA запрещены:
- ✓ `! @ & • $ # ? , . (точка) { } ( ) [ ] = -^ % / ~ < > : ;`
- ✓ Не допускаются пробелы в именах.
- ✓ Длина имени не должна превышать 255 символов (40 символов— для форм и элементов управления).
- ✓ Имя должно не совпадать ни с каким ключевым словом, функцией или оператором VBA.
- ✓ В рамках одной и той же области видимости одно и то же имя нельзя использовать для двух разных объектов. Например, все процедуры в модуле должны иметь разные имена. Не допускается, чтобы переменная процедуры и переменная уровня модуля (определенная в разделе `Declarations` модуля) имели одинаковые имена. Однако можно использовать одно и то же имя для разных переменных, если эти переменные локальны и размещаются в разных процедурах.

Если вы попытаетесь ввести имя, нарушающее эти правила, редактор Visual Basic сообщит об этом, как только вы уберете текстовый курсор из соответствующей строки программного кода. Символы в строке станут красными, и на экране возникнет сообщение с так называемым объяснением ошибки (рис. 6.1). Исключение: предупреждение о дублировании имен процедур не появится до тех пор. пока программа не начнет выполняться.

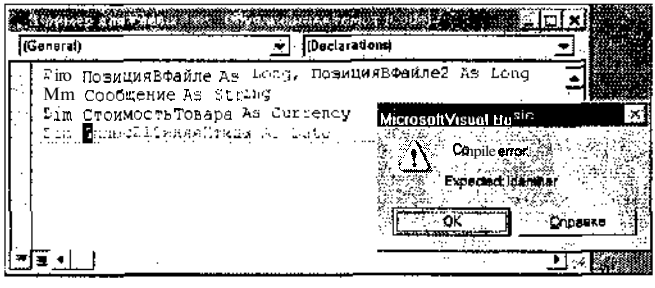


Рис. 6.1. Редактор Visual Basic предупреждает, что введено недопустимое имя

Вот несколько примеров допустимых и недопустимых имен.

В VBA прописные и строчные буквы не различаются, но введенные вами прописные буквы любезно сохраняются. Предположим, что вы объявили переменную, имя которой состоит только из прописных букв, например:

```
Dim LOUDVARIABLE As String
```

Допустимое	Недопустимое
a	ИмяБолее255СимволовНеПоместившееся Здесь...
Go4It	4ScoreAnd7
You_Did_It	WhoDunIt?
СиниеГлаза	Черные Глаза

В программном коде все равно можно будет печатать имя переменной, используя как прописные, так и строчные буквы. Например, в каждом из следующих трех операторов используется одна и та же переменная:

```
loudvariabie = "Ярко-красная"
lOuDvArIaBlE =Ucase(lOUDvARIABLE)
MsgBox(LoudVariable)
```

Конечно, слишком долго наслаждаться альтернативными версиями имени вам не удастся. Редактор Visual Basic запоминает то имя, которое вы задаете в объявлении, и если вы напечатаете имя как-то иначе, оно будет автоматически скорректировано, как только вы перейдете в другую строку.

Соглашения об именах в VBA

В рамках правил, обсуждавшихся в предыдущем разделе, объектам программы можно назначать любые имена. Тем не менее можно значительно облегчить себе жизнь в программировании, если придерживаться определенной логичной схемы выбора имен. По мере того как ваши программы будут становиться длиннее, все труднее будет помнить о типе и назначении каждой конкретной переменной. Поэтому имеет смысл научиться создавать имена, которые смогут кое-что подсказать.

В идеале, нужно выбрать определенный метод и строго его придерживаться. Один такой метод, используемый многими программистами, состоит в том, чтобы имя начиналось с некоторого префикса, соответствующего типу объекта, а за префиксом следовало сокращенное описательное имя, которое начинается с прописной буквы. Например, если при создании программы инвентаризации требуется целая переменная, в которой должен храниться инвентарный номер, то вполне подходящим именем для такой переменной будет `intPartNo`.

В табл. 6.1 приведены префиксы, рекомендуемые для часто используемых объектов VBA. Вместо предложенных вы можете создать свои собственные или использовать их как суффиксы. Главное — непременно их использовать.

В главах 1-5 я не придерживался этих соглашений по той простой причине, что сначала их нужно было объяснить. Но начиная с этой главы я постараюсь следовать тому, что здесь проповедую.

Таблица 6.1. Префиксы, рекомендуемые для имен объектов VBA

Префикс	Тип объекта	Пример
Переменные		
byt	Byte	`bytDaysInMonth`
bool	Boolean	`boolClearedStatus`
int	Integer	`intWeeksOnChart`

Окончание табл. 8.1

Префикс	Тип объекта	Пример
Переменные		
lng	Long integer	lngPopulation
sng	Single	sngRadius
dbl	Double	dblParsecs
cur	Currency	curUnitPrice
str	String	strLastName
date	Date/Time	dateBirthdate
var	Variant	varSerialNumber
obj	Object	objStampCollection
Элементы управления		
txt	Text box (текстовое поле)	txtEnterName
lbl	Label (надпись)	lblAnswerMessage
cmd	Command button (кнопка)	cmdCalculateInterestRate
mnu	Menu (меню)	mnuTools
cmb	Combo box (поле со списком)	cmbToyCategory
fra	Frame (фрейм)	fraHabitat
opt	Option button (переключатель)	optGasolineGrade
chk	Check box (флажок)	chkCaseSensitive
Другие		
bas	Module (модуль)	basTextFormatFunctions
frm	UserForm (пользовательская форма)	frmOptionsDialog

Сделайте программный код красивым

В общем-то, программному коду совсем не обязательно выглядеть красиво — нужно, чтобы он легко читался. В этом разделе предлагается несколько простых рекомендаций по оформлению программного кода для того, чтобы вам было легче расшифровать свой же программный код завтра, через неделю или в следующем году, если это потребуется.

Отступы в программе

Запомните: следует выработать правила использования отступов и неуклонно их придерживаться. Компилятор VBA игнорирует все пробелы в начале строк, так что вы можете смело использовать отступы для наведения порядка. Сравните следующие два фрагмента программного кода и решите, какой из них легче понять:

```
If intA = 27 Then
If txtChooseColor.Text = "Беж" Then
```

```
intA = 33
intB = 0
End If
For Each objCbar In CommandBars
If objCbar.Name = "Моя панель инструментов" Then
If objCbar.Visible = False Then
objCbar.Visible = True
End If
End If
Next objCbar
End If

If intA = 27 Then
     If txtChooseColor.Text = "Беж" Then
          intA = 33
          intB = 0
     End If
     For Each objCbar In CommandBars
          If objCbar.Name = "Моя панель инструментов" Then
               If objCbar.Visible = False Then
                    objCbar.Visible = True
               End If
          End If
     Next objCbar
End If
```

Оба эти фрагмента дают один результат, но отступы во втором фрагменте позволяют сразу сказать для каждого из операторов End If, какой из расположенных выше операторов If...Then будет ему парой. В результате легче отследить путь выполнения программы в зависимости от сложившихся условий.

Правила для отступов

В каких же строках программного кода следует сделать отступ и какой по величине? Нужно установить отступы одного размера для связанных по смыслу операторов, чтобы связь между такими операторами была зрительно очевидной. Конкретнее говоря, операторы, выполняющиеся вместе при каком-то общем условии, должны иметь и одинаковые отступы.

Например, если VBA выполняет операторы внутри конструкции If...Then...Else, Do...Loop или For Next как одну группу, то операторы такой группы должны иметь один и тот же отступ. Вот подходящий пример программного кода:

```
Do While intC <> 20
     intA = intA + 1
     If intA = intB Then
          intA = 5
          intB = 10
     Else
          intA = intB
          intC = 20
     End If
Loop
```

VBA выполняет оператор intA = intA + 1 иструктуру If...Then... Else прикаждом проходе структуры Do...Loop. По этой причине я использовал один и тот же отступ и для оператора intA = intA + 1, и для трех операторов, задающих структуру If...Then...Else

(If. . .Then, Else и End If). Два оператора, следующие непосредственно за оператором If . . .Then, будут выполняться, если только intA = intE, поэтому эти два оператора получают пополнительным отступ, как и два оператора, следующие за Else.

Обратите внимание, *что управляющие структуры* типа Do . . . Loop и If. . .Then. . .Else всегда состоят как минимум из двух операторов (один должен определить начало структуры, а второй — ее конец). Для структуры Do. . .Loop завершающий оператор Loop, а для If. . .Then . . .Else— это End If. Если в рамках структуры есть другие операторы, для них следует использовать тот же отступ. Это позволит ясно различить и саму структуру, и содержащиеся в ней подчиненные операторы. (Управляющие структуры будут подробно рассмотрены в главе 8.)

Как сделать отступ

Для добавления отступа в строке программного кода нажмите клавишу пробела или <Tab>, чтобы сдвинуть текстовый курсор вправо. Можно задать расстояние, на которое клавиша <Tab> сдвигает курсор, если выбрать Tools⇨Options. В диалоговом окне Options (Параметры) на вкладке Editor (Редактор) есть поле Tab Width (Отступ табуляции), где можно впечатать число, соответствующее количеству пробелов. Поскольку пространства в окне редактора Visual Basic не слишком много, лично я установил отступ табуляции, равный 3 пробелам, хотя можно указать любое число от 1 до 32.

Если по каким-то причинам нажимать клавишу табуляции вам не слишком удобно, всегда можно сдвинуть начало строки вправо или влево с помощью предлагаемых VBA кнопок Indent (Добавить отступ) и Outdent (Удалить отступ) соответственно. Эти кнопки применяются не только к отдельным строкам, но и к выделенным фрагментам текста, так что одним щелчком можно добавить или убрать отступ целого блока. Обратите внимание на то, что при использовании этих кнопок положение точки ввода в строке игнорируется — всегда увеличивается или уменьшается отступ сразу всей строки.

Автоматическиеотступы

Чтобы до минимума уменьшить объем выполняемой вами работы, редактор Visual Basic автоматически устанавливает отступ в новой строке, равный отступу в предыдущей. Если в новой строке отступ должен быть меньше, просто нажмите клавишу удаления символа перед курсором (клавишу <—>). Если навязанная помощь вас слишком уж раздражает, автоматическое создание отступов можно отменить на вкладке Editor в диалоговом окне Tools⇨Options.

Свободное пространство - это хорошо

Оставленные между группами родственных операторов пустые строки намекают на важность выделенного таким образом фрагмента программного кода. Кроме того, это уменьшает плотность программы и еще упрощает зрительное восприятие программного кода (рис. 6.2).

Не пользуйтесь прокруткой без необходимости!

Одна строка окна программного кода может содержать примерно 300 символов (наверное, вы удивитесь, если я скажу, что максимальное число символов равно 308), так что при желании можно вместить в одну строку довольно длинный оператор. Однако очевидно и то, что удобнее иметь дело с операторами, полностью видимыми в рамках окна программного кода, когда текст в окне не приходится прокручивать туда-сюда.

Использование символа продолжения строки

Чтобы продолжить оператор на следующую строку, поместите в конце текущей строки символ подчеркивания (_). Например, следующие три строки программного кода в совокупности определяют один оператор:

```
sngWackyNumber = Cos(12 * 57.5 / Sqr(intMyTinyNumber + _
intMyBigNumber) + CustomDataMassage(sngRawInfo, 12) + _
(bytFirstTuesdaylnAugust * curLastPayCheck) + 1)
```

Не забудьте перед символом подчеркивания, который называют еще *символом продолжения строки*, оставить пробел, поскольку без такого пробела вы получите от VBA сообщение об ошибке типа "недопустимый символ" (invalid character).

И еще одно предостережение: следите, чтобы символ подчеркивания не оказался внутри кавычек, определяющих строку текста (если, конечно, вы не собираетесь включить символ подчеркивания в строку). Подробную информацию об использовании строк символов, включая и то, как можно продолжить их на новую строку, вы найдете в главе 7.

Неиспользуйте операторы в несколько строк

Хотя операторы в несколько строк и предпочтительнее длинных операторов в одну строку, не умещающихся в рамках окна программного кода, первые тоже нельзя назвать идеальными. Если VBA и не испытывает неудобств при обработке таких строк, то большинство людей воспринимают операторы в несколько строк как дополнительный источник путаницы.

Если окно программного кода не совсем уж микроскопическое, практически всегда можно разделить длинный оператор на несколько достаточно коротких, в совокупности выполняющих ту же работу, что и исходный. Возможно, для этого потребуется создать несколько дополнительных переменных, в которых будут храниться результаты промежуточных вычислений, но это не слишком большая цена за достижение лучшей прозрачности программного кода. Например, предыдущий пример программного кода можно переписать так:

```
sngTemp1 = Sqr(intMyTinyNumber + intMyBigNumber)
sngTemp2 = 12 * 57.5 / sngTemp1
sngTemp3 = CustomDataMassage(sngRawInfo, 12)
sngTemp4 = bytFirstTuesdaylnAugust * curLastPayCheck
sngWackyNumber = Cos(sngTemp2 + sngTemp3 + sngTemp4 + 1)
```

Этот фрагмент программного кода длиннее и использует несколько дополнительных переменных, но зато за представленными в этом фрагменте вычислениями легче следить. Кроме того, если результат вычислений окажется неправильным, для локализации ошибки можно будет проследить за каждым шагом вычисления в отдельности — ясно, что подобный *пошаговый анализ* невозможен, когда результат вычисления является результатом выполнения одного длинного оператора. (Подробно приемы пошагового выполнения программного кода разбираются в главе 9.)

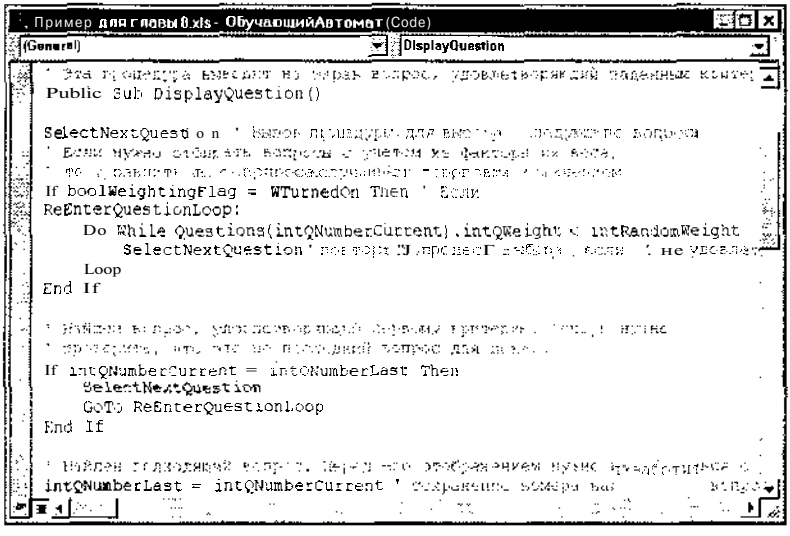
Замечания о комментариях

Как и любой другой серьезный язык программирования, VBA позволяет добавлять в программный код *комментарии,* которые переведут программный код на нормальный человеческий язык, и объяснить назначение каждого оператора или группы операторов. Если уж на то пошло, то с помощью комментариев можно добавить в текст программы и телефонные номера, и рецепты, и уверения в совершеннейшем почтении.

Ваши комментарии компилятор VBA будет полностью игнорировать. Они живут только в текстовом файле, содержимое которого представлено в окне программного кода, но не в откомпилированной программе. Комментарии не увеличивают откомпилированную программу ни на бит, а также никоим образом не замедляют ее выполнение. Комментарии не будут стоить вам ничего, кроме мизерного объема дискового пространства, так что используйте их совершенно свободно.

Как создавать комментарии

Комментарий начинается с напечатанного вами апострофа. Все, что напечатано в строке программного кода справа от апострофа, считается комментарием. Для комментария можно выделить отдельную строку, а можно и добавить его в конец строки с активным программным кодом. Пример окна программного кода с множеством комментариев, добавленных после операторов, показан на рис. 6.2.



flf/c.ff.2*Какиебудуткомментарии?*

Единственное место в окне программного кода, где разместить комментарий невозможно, — это конец строки, в которой используется символ переноса строки (символ подчеркивания). Например, при попытке разместить комментарий так. как это показано в следующем фрагменте программного кода, вы получите сообщение об ошибке:

```
a = b _ ' Это первая из нескольких строк оператора
+ c
```

Чтобы создать комментарий для оператора, занимающего несколько строк, придется разместить комментарий либо в отдельной строке, предшествующей этому оператору, либо в последней строке самого оператора, например:

```
' Можно разместить комментарий здесь
z = x - _
y ' Последняя строка тоже МОЖЕТ содержать комментарий
```

Когда использовать комментарии

Не скупитесь на комментарии. Приучите себя добавлять хотя бы краткие объяснения по поводу каждой отдельной строки программного кода и описывать подробно функциональное назначение групп операторов, При объявлении переменной добавьте комментарий о том, для чего эта переменная создается и где будет использоваться. При объявлении процедуры запишите, что она делает, какие аргументы и для чего использует, какие другие процедуры вызывает. Приучите себя печатать комментарии всегда, как только наступает затишье в творческом процессе создания программного кода.

Простите уж мою настойчивость, но это действительно важно: во время активной работы с конкретным проектом логика создаваемого программного кода может казаться прозрачной и очевидной. Но это прекрасное чувство ясности вскоре растает — и растает без следа. Когда ваше внимание переключится на что-то другое, уже через несколько дней только комментарии избавят вас от исключительно трудоемкой работы по переводу того, что было сделано вами же, на язык, снова понятный вам. Еще важнее комментарии при работе над проектом, в разработке которого принимают участие другие люди, — в этом случае комментарии просто необходимы для того, чтобы проект вообще мог развиваться.

Комментарии можно также использовать для удаления из потока выполнения программы временно ненужных операторов вместо непосредственного их удаления из программы. Это особенно удобно, когда требуется выяснить эффективность нескольких вариантов решений некоторой проблемы или для того, чтобы временно утихомирить фрагмент программного кода, содержащий ошибку, пока вы отлаживаете другую часть модуля.

Пространные комментарии

Чтобы разместить в программе многословный комментарий в несколько строк, необходимо поставить по апострофу в каждой строке, занятой этим комментарием. К счастью, в VBA предусмотрена кнопка, с помощью которой можно превращать в комментарии целые блоки текста сразу.

Чтобы вставить в программный код несколько строк комментария, напечатайте текст комментария, как будто это происходит в обычном текстовом процессоре, игнорируя все сообщения редактора Visual Basic об ошибках, появляющиеся на экране каждый раз. когда вы будете нажимать клавишу <Enter> для перехода на новую строку. Закончив печатать, выделите весь блок текста, которому предназначено стать комментарием, и щелкните на кнопке Comment Block (Добавить в блоке знаки комментария). Редактор Visual Basic добавит необходимый апостроф в начало каждой строки в выделенном блоке.

Имеется также возможность "раскомментировать" блок текста, в начале строк которого проставлены апострофы. Для этого выберите блок текста и щелкните на кнопке Uncomment Block (Снять в блоке знаки комментария}.

С помощью кнопок Comment Block и Uncomment Block можно быстро выключать из потока выполнения программы и снова включать в поток любой блок программного кода. Превращение в комментарий блока программного кода, который пока работает неправильно, очень удобно, когда нужно сначала отладить другую часть программы.

Одно небольшое замечание по поводу тонкостей работы этих кнопок. Если строка уже начинается с апострофа, кнопка Comment Block добавит в начало строки еще один апостроф и т.д. Точно так же, каждый щелчок на кнопке Uncomment Block убирает только один апостроф. А это значит, что если вы с помощью этих кнопок сначала превратите фрагмент выполняемого программного кода в комментарий, а затем снимете знаки комментария, вы не потеряете реальных комментариев, которые могли присутствовать в данном фрагменте программы до всех этих манипуляций.

Убежище **Rem** для комментариев

Ключевое слово `Rem`, которое сохранилось еще со времен самых первых версий языка BASIC, выполняет в VBA ту же самую задачу, что и апостроф, — указывает на начало комментария. (*Rem* —сокращение от английского *remark* (примечание), как нетрудно догадаться.) Если буквы вам нравятся больше, чем знаки пунктуации, используйте Rem вместо апострофов или комбинируйте эти два признака комментария по своему усмотрению.

Между прочим, эти два признака комментария все-таки имеют одно важное различие в использовании. Чтобы использовать ключевое слово Rem в конце строки, содержащей активный программный код, после этого программного кода перед Rem необходимо напечатать двоеточие (:):

```
Rem Здесь начало комментария задается ключевым словом Rem
a = b + c: Rem Видите двоеточие перед Rem?
```

Глава 7

Хранение и обработка информации

В этой главе...

- Использование переменных как именованных контейнеров для хранения самых различных видов данных
- Объявление переменных в программном коде
- Использование именованных констант
- Представление значений с помощью выражений
- Организация вычислений с помощью операторов VBA
- Практическое использование различных типов данных для переменных и констант

Чтобы максимально раскрыть потенциал VBA, программам требуются *переменные* для хранения информации. Переменные позволяют создавать программы, изменяющие свое поведение в зависимости от условий. В этой главе достаточно подробно описаны приемы работы с переменными в VBA-программах. Особое внимание уделяется типам данных, которые могут храниться в переменных. Подобные вопросы рассматриваются также для *констант* (представляющих постоянные значения) и *операторов* (состоящих из знаков операций и ключевых слов и предназначенных для вычисления новых значений на основании уже известных).

Достаточно обширный материал этой главы все же не исчерпывает предмет обсуждения полностью. В главе 12 показано, как использовать переменные для работы с объектами, а в главе 13 дополнительно обсуждаются переменные в связи с массивами и пользовательскими типами данных.

Для всех примеров, представленных в этой главе, на сервере издательства "Диалектика" имеются вполне работоспособные тексты соответствующих процедур. Они доступны через Internet по адресу www. d i a l e k t i k a . com.

Работа с переменными

В сущности, переменная — это идентификационный ярлык для некоторого хранящегося в программе фрагмента информации.

В некотором смысле переменная похожа на жетон из театрального гардероба. Вы приходите в театр и сдаете свою шляпу в гардероб. Гардеробщик куда-то уносит вашу шляпу — вас не интересует куда, поскольку вы уверены, что она никуда не денется. Вместо шляпы вы получаете жетончик с номером.

После спектакля вы отдаете гардеробщику жетон и получаете свою шляпу. На следующий день тот же жетон выдадут в обмен на чью-то другую шляпу.

Вы говорите, что у вас нет шляпы? Так в ответ можно сказать, что до сих пор у вас не было и переменных.

С помощью переменных вы получаете возможность доступа по имени сразу к целому куску информации, вместо того чтобы работать с конкретными числами или символами. Более того, при этом не требуется знать, где и в какой форме хранятся нужные вам данные, — вам нужно знать только имя переменной. Наконец, информацию, которая хранится в переменной, можно в любое время заменить.

Именно последнее дает возможность создавать программы, автоматически реагирующие на изменение условий, поскольку программа может при необходимости проверить информацию, содержащуюся в переменной, и выяснить, удовлетворяет ли эта информация заданным вами критериям. Если критерии удовлетворяются, программа выполняет одни операторы, а если нет, то другие.

Строго говоря, переменная (т.е. контейнер для информации) и хранящаяся в этой переменной информация — это не одно и то же. Однако, наверное, найдется не слишком много людей, которые часто употребляют фразы типа "информация, представляемая переменной x". Вполне привычно ссылаться на информацию просто по имени переменной, имея при этом в виду, что эта информация может меняться.

Объявление переменных

Как уже говорилось в главе 6, *объявление* — это VBA-оператор, с помощью которого вы сообщаете компилятору о своих намерениях использовать именованный объект и явно описываете тип этого объекта. Операторы объявлений чаще всего используются для объявления переменных. Вот несколько примеров объявления переменных:

```
Dim varAnyOldVariable
Private intIntegerVariable As Integer
Static strNewYearsResolution As String
```

Обычно в объявлениях используется ключевое слово Dim, возникшее как сокращение от слова *Dimension* (размер), но как глагол в том смысле, что оператор Dim задает в VBA пространство для хранения данных, на которое будет ссылаться переменная. Точно так же для объявления переменных можно использовать ключевые слова `Private`, `Public` и `Static`, тем самым сразу объявляя и *область видимости* таких переменных (подробности ниже, в разделе "Задание области видимости переменной", а также в главе 6).

После объявления переменной ее можно использовать в других частях программы. Как правило, сначала с помощью оператора присваивания в переменную помещается некоторая информация (подробно об этом — ниже, в разделе "Размещение информации в переменных"). Потом эту информацию можно будет извлечь (используя имя переменной) в любой другой части программы.

Где объявлять переменные

Объявить переменную можно в двух частях программы:

- ✓ в разделе объявлений модуля (на уровне модуля),
- ✓ внутри любой процедуры (на уровне процедуры).

От места объявления зависит, какие процедуры смогут получить доступ к переменной, т.е. от места объявления переменной зависит ее область видимости. Если переменная объявляется на уровне модуля, использовать переменную сможет любая процедура в данном модуле. Если переменная объявляется внутри некоторой процедуры, переменной можно будет пользоваться

только внутри этой процедуры. Для назначения области видимости можно использовать также ключевые слова Private, Public и Static. (Области видимости мы уже обсуждали в главе 6 и еще остановимся на них ниже, в разделе "Задание области видимости переменной",)

Хотя VBA и не возражает против того, чтобы между объявлениями в процедуре размещались выполняемые операторы, программный код будет восприниматься проще, если поместить все объявления в самое начало процедуры. Вот образец правильного размещения операторов внутри процедуры:

```
Public Sub VariableProcedures()
      Dim strChildsName As String
      Dim intToyCount As Integer
      Dim curAllcwance As Currency

      strChildsName = InputBox("Введите имя ребенка:")
' в следующих двух строках CountToys И NewAllowance
' представляют процедуры типа Function, определенные
' где-то в другой части программы
      intToyCount = CountToys(strChildsName)
      curAllowance = NewAllowance(strChildsName, intToyCount)
End Sub
```

Когда объявлять переменные

Все эксперты, как один, рекомендуют объявлять переменные заранее, а не там, где они должны первый раз использоваться в программе. Один из очевидных аргументов в пользу этого — упрощение программного кода. Другие аргументы начинают работать, когда используется директива Option Explicit для компилятора, которая будет обсуждаться ниже, в разделе "Явные объявления приветствуются".

Правда, по умолчанию VBA не запрещает использование переменных без предварительного их объявления. Если где-нибудь в программе вы напечатаете A = 7, VBA добросовестно создаст переменную с именем *A* и поместит в нее значение 7. Переменные, которые вы создаете без предварительного объявления, автоматически получают тип Variant, который будет рассмотрен в следующем разделе.

Выбор и использование типов данных

В объявлении переменной можно просто указать ее имя. Следующий оператор дает указание VBA выделить пространство для переменной с именем varIable, но не указывает, информацию какого типа будет эта переменная хранить:

```
Dim variable
```

Альтернативой будет явное объявление *типа данных* переменной. Оператор

```
Dim sngMyOldSocks As Single
```

объявляет sngMyOldSocks как переменную типа Single, что означает переменную, хранящую не слишком большое число с плавающей запятой, — что-то вроде чисел, использующихся для представления физических величин, с десятичной запятой и экспонентой, как, например, 6,02 X 10^{23}.

VBA предлагает целый ряд других типов данных для переменных, включая исключительно удобные типы Date (Дата) и Currency (Денежный). Внимательно ознакомьтесь со всеми доступными типами данных, поскольку их правильное использование даст ключ к созда-

нию правильно работающих программ. В табл. 7.1 приводятся основные факты, касающиеся типов данных, а дополнительные сведения и обсуждение приемов работы с конкретными типами данных вы найдете ниже, в разделе "Дополнительные сведения о типах данных".

Таблица 7.1. Типы данных VBA

Тип данных	Содержимое соответствующей переменной	Диапазон допустимых значений
Boolean	Логическое Истина или Ложь	Истина (–1) или Ложь (0)
Byte	Достаточно малое целое число	От 0 до 255
Integer	Не слишком большое целое число	От -32768 до 32767
Long	Большое целое число	От –2147483648 до 2147483647
Single	Значение одинарной точности с плавающей запятой	От –3,402823E38 до –1,401298E-45 для отрицательных значений и от 1,401298E-45 до 3,402823E38 для положительных
Double	Значение двойной точности с плавающей запятой	От – 1,79769313486232E308 до - 4,94065645841247E–324 для отрицательных значений и от 4,94065645841247E–324 до 1,79769313486232E308 для положительных
Currency	Большое число, для которого выделено 19 позиций, включая фиксированные четыре позиции после запятой	От – 922337203685477,5808 до 922337203685477,5807
Decimal	Еще большее число, всего 29 позиций, из них до 28 позиций включительно для значения дробной части числа	Допустимый диапазон зависит от числа знаков после запятой, например, ±79228162514264337593543950335 для чисел без дробной части или ±7,9228162514264337593543950335 для чисел с 28 знаками после запятой
Date	Дата и время	От 1 января 100 г. До 31 декабря 9999 г.
Object	Объект VBA	Ссылка на любой объект
String (переменной длины)	Последовательнхть переменной длины, состоящая из символов	Строковая переменная переменной длины может содержать от 0 до примерно двух миллиардов символов
String (фиксированной длины)	Последовательность заданной длины, состоящая из символов	Строковая переменная фиксированной длины может содержать от 0 до примерно 65400 символов
Variant	Любое из допустимых данных	Зависит от типа данных, содержащихся в переменной, в соответствии с вышеприведенными описаниями
Пользовательский (требует использования оператора Type)	Группа переменных, используемых вместе как единое целое	Для каждой из переменных в группе, зависит от ее типа в соответствии с вышеприведенными описаниями

Использование конкретного типа данных по сравнению с типом Variant: за и против

Одни считают, что указание в объявлении переменной ее конкретного типа данных (т.е. типа, отличного от Variant) — это хорошо, другие же считают, что это плохо. Объявление конкретного типа данных переменной помогает избежать одних ошибок, но обеспечивает массу возможностей для появления других. Такой подход способствует уменьшению и ускорению выполнения программ, но в большинстве реальных случаев эти усовершенствования, наверное, не будут слишком важны.

Вот некоторые обстоятельства, способствующие уменьшению числа ошибок. Если при объявлении переменной задается конкретный тип данных, VBA не позволит вам поместить в эту переменную данные других типов. Например, если переменная объявляется оператором

```
Dim dateAnniversary As Date
```

то VBA выведет на экран сообщение об ошибке, когда вы попытаетесь присвоить этой переменной текстовое значение, как здесь:

```
dateAnniversary = "Поездка на Фиджи"  ' Ошибка
```

Сообщения об ошибках, наверное, будут раздражать, но все же это лучше, чем позволить программе работать с ошибочными данными.

Сообщение об ошибке не появилось бы, если dateAnniversary была объявлена как переменная типа Variant, т.е. как переменная, допускающая любой тип данных. Тип данных Variant можно объявить точно так же, как и любой другой, например:

```
Dim varToolSuite As Variant
```

Но поскольку тип данных Variant подразумевается по умолчанию, его явное указание можно опустить, оставив в объявлении только имя переменной:

```
Dim ToolSuite
```

Что же касается вопросов оптимизации, то, указывая конкретный тип, вы имеете возможность выбрать при этом тот тип данных, который обеспечивает в точности необходимый минимум для хранения соответствующей информации. Переменная типа Variant занимает больше памяти, чем переменная любого другого типа, да и доступ к такой переменной осуществляется медленнее. Однако такие аргументы начинают играть роль в программах с очень и очень большим числом переменных. Зато объявив все переменные как переменные типа Variant, вы получаете возможность избежать многих ошибок, сделать программный код более гибким, а также сделать более простой модификацию такого программного кода. Я здесь не могу обсуждать этот вопрос подробнее, но вы должны знать, что некоторые очень хорошие программисты настоятельно рекомендуют программировать, используя исключительно переменные типа Variant.

Выберите подходящую для себя стратегию — объявление конкретного типа данных для каждой переменной или использование типа Variant для всех переменных — и придерживайтесь одной стратегии по крайней мере в рамках одной программы.

Явные объявления приветствуются

Независимо от того, решите вы назначать конкретные типы данных переменным или нет, каждую переменную перед использованием следует объявить. Как серьезный программист, ы можете сделать так, чтобы VBA заставлял вас объявлять переменные. Если в раздел объявлений модуля поместить оператор

```
Option Explicit
```

то при попытке использования предварительно не объявленной переменной VBA будет сообщать об ошибке. После такого своевременного предупреждения вы сможете добавить необходимое объявление в подходящем месте программы.

Такая *требовательность* VBA в деле явного объявления переменных дает и более ощутимые выгоды: тем самым предотвращается в перспективе вероятность появления в программе серьезных ошибок из-за банальных опечаток.

Представьте себе следующую ситуацию. Вы решили не использовать оператор `Option Explicit` в своем модуле. Но вот где-то в глубине процедур в результате случайной незамеченной неточности движения пальцев вы ошибаетесь при печатании имени одной из своих переменных. Когда компилятор VBA дойдет до строки с этой опечаткой, он подумает, что здесь вы просто вводите в оборот новую переменную, и любезно создаст ее для вас. В результате ваша программа может и будет выполняться, но не удивляйтесь, если вдруг выводимый на экран текст окажется ярко-красным или программа сообщит, что население Земли достигло 12 человек.

Наверное, к этому моменту я вас уже убедил, что оператор `Option Explicit` должен присутствовать в каждом создаваемым вами модуле. Но если это так, то зачем этот оператор печатать каждый раз вручную? Это делать совсем не обязательно. Выберите Tools⇨Options из меню и в появившемся диалоговом окне Options (Параметры) на вкладке Editor (Редактор) установите флажок Require Variable Declaration (Требовать объявление переменных). После этого редактор Visual Basic будет автоматически вставлять оператор `Option Explicit` в каждый *новый* модуль. В *старые* модули, где такого оператора не было, вам придется впечатать его самим.

Задание области видимости переменной

Область видимости переменной (т.е. та часть программы, в рамках которой данная переменная доступна) зависит от следующих двух взаимосвязанных факторов:

- ✓ места объявления переменной (либо внутри процедуры, либо в разделе объявлений модуля; см. выше раздел "Где объявлять переменные");
- ✓ ключевого слова, использованного в объявлении переменной (`Dim`, `Public`, `Private` или `Static`).

Если переменная объявлена внутри процедуры с помощью ключевого слова Dim, то такую переменную можно использовать только внутри этой процедуры. В других частях программы VBA переменную не распознает. А вот переменные, объявленные с помощью Dim в разделе объявлений модуля, будут доступны из любой части модуля, но не из других модулей.

По умолчанию переменные локальны

Ключевое слово `Private` работает точно так же, как и `Dim`. Например, следующие два оператора объявления функционально идентичны:

```
Private strLouie As String

Dim strLouie As String
```

Ввиду того, что ключевые слова `Private` и Dim работают одинаково, вы можете вообще не использовать, например, `Private`. А можно использовать только `Private`, если вы хотите, чтобы это ключевое слово напоминало о том, что переменная достижима только в рамках данной процедуры или модуля.

Глобальное знание

Объявление переменной с помощью ключевого слова Public делает эту переменную доступной из любой части проекта. Например:

```
Public intUnclassifield As Integer
```

Правда, такую особую силу ключевое слово Public имеет только тогда, когда вы объявляете переменную в разделе Declarations (Объявления) модуля. VBA не будет возражать и против использования Public внутри процедуры, но таким образом объявленные переменные все равно останутся невидимыми вне процедур, в которых заключаются объявления.

Статическое состояние

Ключевое слово Static в объявлении переменной следует использовать тогда, когда вы хотите, чтобы переменная оставалась в памяти, — для того чтобы использовать ее значение — даже когда процедура завершила свою работу. В следующем примере переменная intLastingVariable используется как счетчик числа вызовов данной процедуры:

```
Sub TransientProcedure()
Dim strTransientVariable As String
Static intLastingVariable As Integer
strTransientVariable = Format(Now(), "Medium Time")
intLastingVariable = intLastingVariable + 1
MsgBox "Сейчас " & strTransientVariable & ". " & _
"Эта процедура выполнялась " &
intLastingVariable & " раз(а)."
End Sub
```

В этом примере оператор intLastingVariable = intLastingVariable + 1 увеличивает значение переменной на единицу при каждом выполнении процедуры. Если переменную intLastingVariable объявить ключевым словом Dim, а не Static, то в начале каждого выполнения процедуры эта переменная получала бы нулевое значение, что лишало бы процедуру всякого смысла вообще.

Объявлять переменные ключевым словом Static (статические переменные) можно только внутри процедур. Если вы хотите, чтобы все переменные в некоторой процедуре сохраняли свои значения (даже когда процедура не выполняется), поместите ключевое слово Staticв объявление самой процедуры непосредственно перед ключевым словом, определяющим тип процедуры (т.е. перед Sub или Function), например;

```
Private Static Sub DoItAll()

Static Function DontDoVeryMuch(intTimeToWaste As Integer)
```

Обратите внимание, если в объявлении процедуры присутствует задающее область видимости ключевое слово Private (или Public), то Static идет после него.

Объявление нескольких переменных в одной строке

В одной строке программного кода можно объявить и несколько переменных. Ключевое слово Dim печатается при этом только один раз, а переменные разделяются запятыми.

Но не забывайте указать тип данных для *каждой* из переменных в объявлении, даже если все эти переменные одного типа. Вот пример правильно оформленного оператора этого типа:

```
Dim intA As Integer, intD As Integer, ir.tL As Integer
```

Смешивать различные типы данных тоже разрешается:

```
Dim curNetWorth As Currency, datSecondTuesday As Date
```

При объявлении нескольких переменных в одной строке возрастает вероятность случайного пропуска объявления типа данных. Для каждой из переменных в строке с пропущенным типом данных автоматически будет выбран тип `Variant`. Например, при обработке оператора

```
Dim strX, strY, strZ As String
```

`strX` и `strY` интерпретируются как переменные типа `Variant`, а не как строковые переменные.

Размещение информации в переменных

После того как переменная объявлена, первым шагом в ее использовании обычно бывает наполнение ее информацией (первое размещение информации в переменной называется *инициализацией* переменной). Чтобы поместить информацию в переменную, нужно *присвоить* значение информации этой переменной. Как только понадобится, можно заменить данные, хранящиеся в переменной, присвоив ей другое значение.

Присваивание значений

Значения переменным присваиваются с помощью обыкновенного знака равенства. Например, чтобы поместить число 3 в переменную с именем `intC`, напечатайте

```
intC = 3
```

В VBA *оператор присваивания* представляет собой связанную знаком равенства конструкцию, с переменной слева от него и выражением, определяющим значение переменной, справа. В приведенном выше примере выражение представляет собой просто число 3. Такие явно указанные значения называются *буквальными* значениями. (Выражения будут обсуждаться ниже, в разделе "Выражен-и-я".)

Взгляните на следующий оператор присваивания:

```
strQuote = "Не спрашивай, что я сделал для страны - " & _
    "спроси о том, что страна сделала для меня."
```

В данном случае оператор присваивает текст справа от знака равенства переменной `strQuote`. Как и раньше, информация во входящем в этот оператор выражении представляет собой буквальные значения — реальный текст, помещаемый в переменную. Но оператор разбит на две строки, поэтому и текстовое выражение разбито на две отдельные строки. Знак `&` (амперсанд) дает указание VBA соединить эти строки вместе.

Здесь я просто пытаюсь показать, что выражения часто имеют несколько компонентов. Но независимо от того, из какого числа компонентов складывается выражение, VBA сначала вычисляет результирующее значение, а уж *затем* присваивает это значение переменной.

Ясно, что до момента реального выполнения оператора присваивания его утверждение не является фактом. В математике равенством 2+2=4 на самом деле утверждается, что значение слева от знака равенства *равно* значению справа. В VBA оператор присваивания *заставляет* переменную быть равной значению выражения. В дальнейшем новый оператор присваивания может изменить значение этой переменной в любое время.

Использование переменных в операторах присваивания

Можно присваивать переменным не только буквальные значения — точно так же можно присваивать и значения, определяемые другими переменными. В операторе

```
curSalePrice = curCost * sngMargin
```

переменной curSalePrice присваивается значение, равное произведению переменных curCost и sngMargin. В данном случае выражение состоит из переменных, а не из буквальных значений. VBA вычисляет результат на основе значений, хранящихся в этих переменных. (Кстати, в полном соответствии с рекомендованными в главе 6 соглашениями по выбору имен, curSalePrice и curCost являются переменными типа Currency.)

Использование функций в операторах присваивания

Поскольку функции и создаваемые вами процедуры типа Function возвращают значения. их тоже можно использовать в операторах присваивания. Например:

```
strFavorite = InputBox("Кто сегодня фаворит?")
```

В данном случае используется функция VBA InputBox, которая отображает на экране небольшое диалоговое окно с указанным сообщением и полем для ввода ответа пользователем. Этот ответ и будет возвращаемым функцией значением, представляющим собой строку текста (функция InputBox рассматривается подробно в главе 11).

Выражен-и-я

Выражение представляет собой часть оператора VBA. которая в результате ее вычисления дает некоторое значение (этим значением может быть число, строка текста или ссылка на объект). Выражение может состоять из одного или нескольких следующих элементов в любой их комбинации:

- ✓ переменные (например, bytMonth или boolWinter);
- ✓ буквальные значения (например, 1234 или Это только пример);
- ✓ константы (они предназначены для хранения буквальных значений и рассматриваются в разделе "Работа с постоянными значениями");
- ✓ функции VBA (например, InputBox () или Sqr ());
- ✓ процедуры типа Function.

Если в выражении не меньше двух таких элементов, то они должны быть связаны либо знаками операций (к последним относится, например, знак +), либо функциями и процедурами типа Function, вложенными внутрь других функций и процедур типа Function.

Обратите внимание на то, что если выражение состоит из нескольких компонентов, то каждый из них тоже будет выражением со своим собственным значением.

Что содержит переменная перед тем, как ей присваивается значение?

Во время выполнения процедуры VBA выделяет для каждой переменной из этой процедуры пространство в памяти и приписывает переменной начальное значение, означающее, что в переменной "ничего не хранится". Чаще всего перед использованием переменной в выражениях вы присваиваете ей нужное вам значение, но вполне допустимо, а порой и выгодно, использовать переменную до того, как вы будете уверены, что она содержит заданное вами значение,

Предположим, что в программе есть процедура, которая должна выполняться только при определенных условиях. Допустим также, что в этой процедуре должно присваиваться значение одной из переменных программы. В такой ситуации, если в другой процедуре нужно выяснить, выполнялась ли первая процедура, то это можно сделать, выяснив, хранится ли в данной переменной значение.

Вот какие значения хранятся в переменных перед тем, как вы им присваиваете свои значения.

Тип данных	Начальное значение
Все числовые типы данных	0
`string` (переменной длины)	Строка нулевой длины (`""`)
`String` (фиксированной длины)	Строка указанной длины, состоящая из символов, ASCII-код которых равен 0 (эти символы невидимы на экране)
`Variant`	Пусто (специальное значение, указывающее на то, что переменная этого типа не хранит никакого значения)
`Object`	Ничто (специальное значение, указывающее на то, что переменной не присвоено никакой ссылки на объект)

Работа с постоянными значениями

Если в программе есть значения, которые *не должны* меняться, то для представления таких значений не обязательно создавать переменные. Конечно, в любой процедуре для представления этих значений можно использовать и буквальные значения, но объявление представляющих эти значения *констант*, как правило, предпочтительнее.

Объявление констант

Для объявления констант используется ключевое слово `Const`:

```
Const cstrPetsName As String = "Пушок"
Const cdateTargetDate As Date = #5/26/2000#
Const cboolUp As Boolean = True
```

Все вполне аналогично объявлению переменных, за исключением того, что при объявлении константы необходимо указать ее значение. При объявлении констант можно указывать те же типы данных, что и для переменных, за исключением типа `Object`, пользовательского типа данных и пока что типа `Decimal` (см. табл. 7.1).

Обратите внимание, что имена объявленных мною констант начинаются со строчной буквы "c" (означающей сокращение от *constant* — константа). Таким образом, по имени сразу видно, что это константа, а не переменная. Если вас этот префикс не устраивает, выберите для своих констант другой, более подходящий вам.

Вы можете последовать примеру VBA и Visual Basic и выбрать префикс, соответствующий вашему имени, или префикс, соответствующий имени вашего VBA-проекта. VBA и Visual Basic идентифицируют константы с помощью префикса `vb` — например `vbBlue` (константа, представляющая числовой код для синего цвета, а именно 16711680) или `vbKeyTab` (константа, представляющая числовой код клавиши <Tab>, а именно 9). VBA-приложения часто идентифицируют свои константы с помощью префикса, специфического для этого приложения, например `xlBarStacked` (константа, представляющая числовой код для линейчатой диаграммы в Excel, а именно 58).

Кстати, константы, определенные VBA или VBA-приложением, тоже можно использовать в своих программах. Информацию об этих зарезервированных константах можно получить из справочной системы или в окне обозревателя объектов (см. главу 5).

Плоды использования констант

Объявив константу, вы получаете возможность использовать ее везде, где в программе потребуется представленное именем этой константы буквальное значение. Скажем, вы создаете программу, которая должна определять зарплату работника в зависимости от размера его обуви. В таком случае основная часть программного кода может выглядеть следующим образом:

```
If bytРазмерОбуви > 45 Then
    curЗарплатаДжо = 75000
End If
```

Основным недостатком такого подхода является то, что при этом величина зарплаты определяется в программе "жестко". Если стоимость жизни возрастет и поэтому потребуется увеличить зарплату, вам придется найти соответствующее значение в программе и увеличить его. А если значение использовано в программе несколько раз, придется найти и изменить каждое из них. При этом возрастает риск опечатки, способной повлечь за собой неправильную работу программы.

Вот тот же фрагмент программного кода, заново переписанный для работы с подходящей константой:

```
Const ccurСамаяБольшаяЗарплата As Currency = 75000
...
If bytРазмерОбуви > 45 Then
    curЗарплатаДжо = ccurСамаяБольшаяЗарплата
End If
```

При таком подходе вы легко найдете объявление константы в самом начале модуля или процедуры, чтобы изменить значение на новое. В результате это новое значение заменит старое во всех частях программы, где используется константа. Кроме того, программный код будет легче понять: вместо того, чтобы спрашивать себя "Что это за число 75000?", вы сможете с первого взгляда уверенно сказать, что Джо будет получать самую большую зарплату, если он носит обувь 46 размера.

Вместо константы *можно* было бы использовать переменную. Но переменные занимают больше места в памяти, а также, что на самом деле важнее, при использовании переменной вместо константы появляется риск случайного изменения в программе значения переменной, которое в данном случае меняться не должно.

Использование констант для атрибутов

Константы очень удобны при работе с группами именованных элементов типа дней недели (понедельник, вторник и т.д.) или характеристик типа вкусовых признаков (сладкий, соленый, кислый, горький). Вместо того чтобы оперировать в программе строками с названиями этих элементов, проще назначить каждому элементу' некоторое число и объявить константу с именем, соответствующим названию элемента, равную этому числу. После этого в программе можно использовать вместо чисел названия. Вот пример программного кода, в котором используется такой подход:

```
Const cbytСладкий = 1, cbytСоленый = 2
Const cbytКислый = 3, cbytГорький = 4
```

```
Do While intBKyc = cbytКислый
ДобавитьСладкого
intBKyc = ПробаВкуса()
Loop
```

Знакиопераций

В VBA *операция* представляет собой либо специальный символ, либо ключевое слово в выражении, комбинирующем два значения (субоператора, если желаете) для получения нового результата.

В следующем выражении операция + (операция сложения) увеличивает на 3 значение переменной intA:

```
intA + 3
```

Значением выражения и будет результат такого сложения. (Не забывайте, что выражение не может выполняться само по себе — оно должно быть частью VBA-оператора, например `intB = intA + 3`. По поводу выражений см. выше раздел "Выражен-и-я".)

VBA разделяет операции на три главные категории — арифметические, логические и операции сравнения, плюс несколько не попадающих в эти категории операций типа *конкатенации* для работы со строками. Чуть позже в этой же главе мы рассмотрим каждую из категорий отдельно.

При использовании со строками текста знак + представляет конкатенацию, т.е. соединение двух строк в одну, а не сложение. Но для строк лучше использовать "настоящий" знак конкатенации — символ &. VBA интерпретирует выражение

```
"Меня зовут " & "Боря"
```

как "Меня зовут Боря".

Кстати, знак операции + можно использовать для тою, чтобы прибавлять к числам текст, если текст состоит только из цифр. Например, в результате выполнения оператора

```
intC = 123 + "456"
```

значением переменной intC будет 579.

А вот пример использования операции сравнения:

```
Tan(sngAngleA) <> 1.4
```

Знак о представляет операцию "не равно"'. С его помощью проверяется, не равны ли два значения в выражении, а результатом его выполнения 5>дст либо True (Истина), либо False (Ложь). Если тангенс sngAngleA *не* равен 1.4, результатом выражения будет True, а иначе результатом будет False.

Получение приоритета

В более сложных выражениях, включающих несколько знаков операций, VBA нужно знать, какие операции выполнять первыми, вторыми, а какие третьими. В выражении

```
intA + intB * inrC
```

два знака операций: + (операция сложения) и * (операция умножения). На русском это выражение читается как "intA плюс intB умножить на intC".

В данном выражении символ * идет вторым, но он имеет *приоритет* перед операцией сложения. Сначала VBA умножит intB на intC, а уж затем добавит полученный результат

к intA. Как можно догадаться из этого примера, VBA следует определенной последовательности выполнения отдельных частей выражения, когда в выражении несколько знаков операций.

Чтобы изменить последовательность операций, порождаемую приоритетами, в выражении используются скобки. Если вы напечатаете

```
(intA + intB) * intC
```

то VBA сначала сложит две переменные, а затем умножит результат на значение переменной intC.

Но каковы правила VBA для порядка выполнения операций в отсутствие скобок? Если выражение содержит знаки операций из двух или нескольких категорий, то VBA выполняет операции из разных категорий в следующем порядке.

1. **Арифметические операции и конкатенация.**
2. **Операции сравнения.**
3. **Логические операции.**

Внутри каждой категории VBA тоже имеет правила порядка выполнения операций. Арифметические операции, сравнение и логические операции осуществляются в порядке, соответствующем их представлению в табл. 7.2. Операции сравнения в VBA выполняются в том порядке, в каком они идут, слева направо. Если в выражении встречается несколько операций одного уровня, они тоже обрабатываются слева направо.

Таблица 7.2. Порядок выполнения операций в VBA (внутри каждой категории операции выполняются втом порядке, в котором приведены)

Знаки операций	Операции
Арифметические	
^	Возведение в степень
–	Отрицание
* или /	Умножение или деление
\	Целочисленное деление
Mod	Вычисление остатка деления
+ или -	Сложение или вычитание
Конкатенация	
&	Конкатенация строк
Сравнения	
=	Равенство
<>	Неравенство (не равно)
<	Меньше
>	Больше
<=	Меньше или равно
>=	Больше или равно

Окончание табл. 7.2

Знаки операций	Операции
Сравнения	
Like	Сравнение строк с заданным образцом
Is	Проверка идентичности ссылок на объекты
Логические	
Not	Логическое НЕТ
And	Логическое И
Or	Логическое ИЛИ
Xor	Логическое Исключительное ИЛИ
Eqv	Логическая эквивалентность
Imp	Логическая импликация

Вычисления с помощью арифметических операторов

В VBA имеется семь знаков арифметических операций (см. табл. 7.2). Что делают четыре из них, вы определите сами с первого взгляда на них — это +, -, * и /. Например, результатом выражения 6/2 будет, конечно же, 3.

А вот остальные три знака арифметических операций не так очевидны. Они выполняют следующие действия.

- ✓ Операция ^ возводит первое значение в выражении в степень, равную второму значению. Например, результатом вычисления выражения

  2 ^ 3

  будет 8 (2^3).

- ✓ Операция \ представляет целочисленное деление, в результате которого всегда получается целое число. VBA просто отбрасывает дробную часть, а не округляет результат. **(Замечание.** Эта операция представлена обратной косой чертой в отличие от обычного деления, которое представлено обычной косой чертой.) Например,

  244 \ 7

  равно 34, что соответствует числу полных недель в периоде времени, составляющем 244 дня.

- ✓ При выполнении операции **Mod** первое значение тоже делится на второе, но в качестве результата возвращается *остаток* от деления. Продолжая тему предыдущего примера, если вы захотите узнать, на сколько дней период времени в 244 дня превышает 34 полностью поместившиеся в нем недели, нужным выражением для вычисления этого будет

  244 Mod 7

  что в результате дает 6. Обратите внимание, что результатом выполнения операции Mod всегда будет целое число. Если вы захотите узнать дробную часть от деления в десятичной форме, используйте выражение типа

  (244 Mod 7) / 7

  Это то же самое, что и 6/7, или примерно 0,857.

Сравнение значений

В VBA используется шесть операций сравнения для сравнения числовых и строковых значений, а также две специальные операции Like (для строк) и Is (для объектов).

Заметьте, что знак равенства в YBA (=) используется в двух разных случаях. В объявлении равенство используется для присваивания значения переменной (см. выше раздел "Размещение информации в переменных"), а в качестве операции сравнения знак равенства определяет, будут ли два значения одинаковыми.

Результатом вычисления выражения, основанного на этих операциях, всегда является либо True (Истина), либо False (Ложь). Например, вот выражение с использованием операции <= (меньше или равно):

```
intX <= 12
```

Если значением intX является 12, то результатом выражения будет False, поскольку 12 не меньше, чем 11.

Чаще всего выражения с операциями сравнения используются в *условных операторах* типа If. . . Then. Условные операторы будут подробно обсуждаться в главе 8, но следующий пример нелишний и здесь:

```
If intX = 2000 Then
    Застолье
End If
```

Здесь с помощью знака операции = выясняется, равны ли значения по обе стороны от него. Если значения равны, то результатом выражения будет True (Истина), иначе False (Ложь); другими словами: "Если значение intX равно 2000, то выполнить процедуру Застолье".

Сравнение строк

Знаки операций сравнения можно использовать для сравнения строк текста и чисел. Выражение

```
"Петуния" = "Нарцисс"
```

в результате дает False — и это очевидно, поскольку эти две строки не одинаковы.

Но есть случаи, когда результат сравнения строк предсказать не так просто. Чтобы получать правильные результаты при сравнении строк, нужно знать, по каким правилам VBA решает, что одна строка "больше", чем другая.

Если вы не потребуете сравнивать иначе, VBA будет использовать метод *двоичного* сравнения. Две строки будут сравниваться на основе числовых кодов входящих в эти строки символов, т.е. кодов, которыми символы представляются в программе. В такой системе кодирования наименьшие числа соответствуют знакам пунктуации, за ними следуют цифры, затем прописные буквы, строчные и, наконец, буквы с акцентами (последнее касается латинской колировки, поскольку в случае кириллической колировки на месте букв с акцентами находятся русские буквы). В этом случае, ввиду того, что коды строчных букв больше кодов прописных букв, выражения "a" > "A" и "a" > "Z", например, оба дают в результате True.

При сравнении строк VBA начинает сравнивать первые символы этих строк. Если первые символы различны, большей строкой будет та, первый символ которой больше. Если же первые символы равны, VBA сравнивает следующие символы, и т.д.

Чтобы при сравнении использовался другой, интуитивно более понятный метод, поместите в раздел объявлений модуля оператор Option Compare Text. В результате этого сравнение строк будет выполняться по алфавиту без учета регистра букв (но все равно будет считаться, что буквы с акцентами имеют более высокие коды, чем соответствующие им буквы без акцентов). *В* этом случае следующие выражения дадут в результате True (Истина):

```
"a" = "A"
"a" < "Z"
```

Сравнение строк разной длины

Но что будет, если придется сравнивать строки неравной длины? Если начало одной строки в точности совпадает с другой, более короткой строкой, то длинная строка считается больше короткой. Например:

```
"В горах, в пещере" > "В горах"
```

дает в результате True (Истина).

Но если две строки различаются не только по длине, но и по содержанию в рамках сравнимой дойны, применяется общий критерий сравнения. Например,

```
"У меня двенадцать" > "У меня двадцать два"
```

дает в результате True (Истина), поскольку буква *е* в слове *двенадцать* больше, чем *а* в слове *двадцать*, несмотря на то, что левая часть выражения короче правой.

Сравнение с помощью Like

Как правило, с помощью Like строка сравнивается не с конкретным набором символов, а с заданным образцом, в котором используются замещающие символы, когда нужно убедиться, что строка попадает (или не попадает) в некоторый класс строк. У меня нет возможности вдаваться здесь в детали, но вы должны знать, что сравнение такого типа в VBA есть и что это довольно мощное средство для обработки текстовых данных.

Операторы сравнения в программном коде

Результат выполнения сравнения можно сохранить в переменной (обычно для этого используется переменная типа Boolean) с помощью стандартного оператора присваивания:

```
boolTheAnswerIs = 5 > 4
```

Поскольку 5 больше, чем 4, результатом сравнения будет True (Истина), а поэтому VBA присвоит переменной boolTheAnswerIs значение True.

Ключевые слова True и False на самом деле встроенные числовые константы VBA, соответственно представляющие значения -1 и 0. Результат сравнения можно присвоить любой числовой переменной.

Операции сравнения часто используются в условных операторах, чтобы принять решение о выполнении той или иной ветви программного кода:

```
If intP <= intQ Then
     ЧтоТоНеТак ' вызоз процедуры обработки ошибок
End If
```

Объединение текста

Знак *конкатенации,* & (амперсанд), обозначает операцию соединения строк в одну. Его можно использовать со строками текста, строковыми переменными или любыми функциями, возвращающими строковые значения. Его можно использовать несколько раз, чтобы строить длинные строки из нескольких строковых значений, как, например, здесь:

```
strA = "Вы ответили " & InputBox("Введите ответ:") & _
     ". Правильным ответом было " & strAnswer & "."
```

В результате выполнения этого оператора strA может содержать, например, "Вы ответили Португалия. Правильным ответом было Испания", или какую-то подобную строку.

При составлении большой строки из маленьких не забывайте добавлять в строку-результат необходимые пробелы и знаки пунктуации.

Дополнительные сведения о типах данных

В этом разделе приводятся рекомендации по поводу того, где и когда использовать различные типа данных VBA. При этом рассматриваются все типы данных, кроме двух. Тип данных Object, хотя и очень полезен, но достаточно сложный, поэтому отдельно обсуждается в главе 12. Пользовательские типы данных в настоящей книге не рассматриваются.

Преобразование типов данных

Типы данных существуют только для удобства программиста — VBA хранит всю свою информацию исключительно в цифровой форме. А поскольку это так, преобразование данных одних типов в другие не слишком большая проблема для VBA.

В VBA есть целый ряд встроенных функций для преобразования одних типов данных в другие. Эти функции рассматриваются в главе 11 и с их помощью вы можете управлять такими преобразованиями. Но важно знать, что всегда, когда это следует из контекста, VBA автоматически конвертирует данные к подходящему типу. Например, если в выражении участвуют строка, состоящая только из цифр, и числовое значение, связанные знаком операции +, то к числовому значению будет прибавлено число, определяемое строкой. Точно так же, если присвоить целой переменной значение с десятичными знаками, то VBA автоматически округлит это значение.

Часто такие автоматические преооразования типов вполне соответствуют тому, что вам нужно. Но основная проблема здесь состоит в том, что даже если вы позаботитесь о явном объявлении типов данных для всех своих переменных, вероятность появления ошибок не исчезает как раз из-за стремления VBA предугадать ваши желания.

Тип Variant

Тип Variant обеспечивает "безразмерный" контейнер для хранения данных. Переменная этого типа может хранить данные любого из допустимых в VBA типов, включая числовые значения, строки, даты и объекты. Более того, одна и та же такая переменная в одной

и той же программе в разные моменты может хранить данные различных типов. Например, следующий фрагмент программного кода вполне допустим (хотя вряд ли продуктивен):

```
Dim varЧтоУгодно As Variant
уагЧтоУгодно = 3
уагЧтоУгодно = "Я полагаю."
уагЧтоУгодно =#12/31/99 11:59:59 PM#
```

VBA не только допускает такой набор операторов, но еще и помнит, данные какого типа вы помещаете в переменную типа Variant. Например, после выполнения последнего из приведенных выше операторов уагЧтоУгодно будет идентифицироваться как переменная типа Variant/Date. Выяснить, данные какого типа хранятся в переменной типа Variant в данный момент, можно с помощью функции VBA TypeName. Например, если бы предыдущий фрагмент программного кода продолжался оператором

```
strVariantType = TypeName (уагЧтоУгодно)
```

то значением переменной strVariantType было бы Date, поскольку данные именно такого типа находятся в уагЧтоУгодно.

Благодаря такой своей гибкости переменные типа Variant очень удобны. Вместо того чтобы заботиться об использовании конкретных типов данных, можно всем переменным назначить тип Variant, и тогда в них легко помешать данные любых типов. Однако такой подход тоже имеет свои недостатки, о которых уже говорилось в разделе "Использование конкретного типа данных по сравнению с типом Variant: за и против".

Но даже если вы назначите конкретные типы данных всем своим переменным, переменные типа Variant не утратят своей роли хотя бы как черновые переменные для простых вычислений в небольших процедурах. А в VBA 5 и VBA 6 такие переменные придется использовать, если вам потребуется максимально допустимая в VBA точность вычислений (подробности — в следующем разделе).

Выбор числового типа данных

Если назначить переменным определенные типы данных, а не тип Variant, и при этом в каждом конкретном случае выбрать наиболее подходящий тип данных, то программа будет выполняться быстрее, будет меньше по объему и, возможно, вероятность ее правильной работы будет выше. Ясно, что переменная должна занимать в памяти объем, достаточный для хранения любого из возможных для нее значений, а все сверх этого — пустая трата памяти.

Например, если в программе требуется одна переменная для хранения информации о дне недели (диапазон возможных значений от 1 до 7) и одна переменная для числа месяца (диапазон 1–31), то вполне достаточными для этого будут переменные типа Byte, позволяющие хранить числовые значения в диапазоне от 0 до 255. Если же для работы вам нужны 365 дней в году, придется использовать переменную типа Integer.

Диапазоны допустимых значений для переменных каждого из допустимых в VBA типов данных были приведены в табл. 7.1. Вот еще несколько советов об использовании некоторых типов данных.

- ✓ Для хранения целых чисел (т.е. не имеющих дробной части) используйте переменные типов Boolean, Byte, Integer или Long.
- ✓ Используйте переменные типов Single или Double, чтобы хранить числа с плавающей запятой и показателем степени, имеющие до 15 значащих цифр. Хотя в этом случае диапазон допустимых значений просто огромен, не упускайте из виду возможность ошибок округления, когда выполняются операции со значениями, сильно отли-

чающимися по порядку. Чтобы присвоить, например, значение $4{,}72\times10^{-22}$ переменной типа Single или Double, используйте следующий формат (когда знак + или - после буквы E пропущен, VBA считает показатель экспоненты положительным):

```
sngFloating = 4.72E-22
```

- ✓ Если требуется еще более высокая точность вычислений, тип данных Currency обеспечит 19 значащих цифр, а тип данных Decimal — все 29 (эти типы данных в своих представлениях чисел не используют экспоненты). Однако в настоящей версии VBA Decimal не является самостоятельным типом данных — это значит, что нельзя объявить переменную типа Decimal, а тип данных Decimal существует только как возможное значение для типа Variant. Чтобы поместить в переменную типа Variant числовое значение не как значение с плавающей запятой, а как значение типа Decimal, используйте в операторе присваивания функцию CDec.
- ✓ Вот пример того, как это реализовать (обратите внимание, что для функции CDec значения с большим количеством значащих цифр придется представлять в виде строк):

```
Dim decvarPi As Variant
decvarPi = CDec("3.14159265358979323846264338327")
```

Если в программе выполняются вычисления с использованием значений некоторых переменных, то тип данных этих переменных должен быть достаточно емким, чтобы хранить *результат* вычисления, даже если этим переменным сам результат и не присваивается. Для примера внимательно ознакомьтесь со следующим фрагментом программного кода:

```
Dim bytByte1 As Byte
Dim bytByte2 As Byte
Dim intInteger As Integer
bytByte1 = 255 ' максимальное значение для типа Byte
bytByte2 = 1
intInteger = bytByte1 + bytByte2 ' ошибка!
```

Здесь, несмотря на то, что переменная intInteger вполне допускает хранение значения, соответствующего результату вычисления, в последней строке этого фрагмента программного кода VBA зафиксирует ошибку переполнения. Чтобы вычисления выполнялись, необходимо в данном случае хотя бы вместо одной из переменных типа Byte использовать переменную типа Integer.

Когда использовать логические переменные

Переменные типа Boolean могут хранить только два значения: True (в числовом представлении это 1) или False (0). Используйте переменные типа Boolean, когда нужно выяснить, какое из двух альтернативных условий имеет место в данный момент. Например, можно использовать переменную boolВключено, которая должна принимать значение True (Истина), когда что бы то ни было, к чему переменная относится, включено, и значение False (Ложь), когда это нечто выключено.

Другим полем для использования переменных типа Boolean является определение констант со значениями True и False. Тогда имена переменных могут быть вполне нейтральными, а вот константы должны явно указывать на существование двух альтернатив для этих переменных. В общем, проще показать пример, чем пытаться объяснить:

```
Dim boolПузо As Boolean
Const Прикрыто As Boolean = True
Const Видно As Boolean = False
If boolПузо = Видно Then
    НемногоПощекотать
End If
```

Вот вам один просто замечательный совет! Чтобы переключить на противоположное значение переменной или свойства объекта типа Boolean, используйте ключевое слово Not. Например, в Word вы можете включить или отключить отображение на экране схемы документа с помощью такой строки программного кода:

```
ActiveWindow.DocumentMap = Not ActiveWindow.DocumentMap
```

Работа с денежными значениями

Главная цель использования типа данньк Currency (Денежный)— получение точного результата. Хотя типы данных Single и Double с плавающей запятой и могут хранить числа с дробной частью — какими обычно бывают денежные значения — вычисления, выполняемые над числами с плавающей запятой, часто порождают небольшие ошибки, а это заставляет сильно нервничать тех, кто занят подсчетом денежных знаков.

Адаптация к местному формату представления денежных величин

Одна из сильных сторон VBA — это возможность автоматической настройки форматов представления дат и денежных величин в соответствии с местными стандартами. В правильно построенную про]рамму вам не придется включать символы типа S, Г или J — VBA добавит именно тот символ, который нужен, основываясь на выборе языка и страны, которые были сделаны с помощью панели управления Язык и стандарты в Windows.

Рассмотрим, например, следующий фрагмент программного кода:

```
Const ccurMoneyTalks As Currency = 5463.72
MsgBox Format(ccurMoneyTalks, "Currency")
```

В рез>льтате выполнения этого фрагмента в США в окне сообщения появится $5,463.72, а та же программа во Франции отобразит 5463,72F. Используемая здесь функция Format достаточно подробно рассматривается в главе 11.

Чтобы отобразить содержимое переменной в виде правильно представленного денежного значения, совсем не обязательно объявлять эту переменную переменной типа Currency. Функция Format с именованным форматом "Currency" замаскирует любое числовое значение под доллары, франки или что-то другое, имеющее хождение в вашем регионе.

Другие возможности использования типа данных Currency

Даже если вы и не собираетесь работать с деньгами, переменные типа Currency будут полезны в следующих случаях:

- ✓ хранение больших чисел. выходящих за границы диапазона, допустимого для целых чисел типа Long;
- ✓ вычисления с большими числами, когда требуется более высокая точность, чем та, которая обеспечивается типами данных с плавающей запятой.

Значения типа Currency могут иметь до 19 значащих цифр, из них 15 — до запятой и 4 — после (положение десятичного разделителя фиксировано).

Работа с датами

Тип данных Date используется для удобства при работе со значениями дат и времени. Где-то в своих глубинах VBA кодирует дату и время в виде некоторого числа вроде 35692,9201273148. которое, очевидно, совершенно ничего не говорит большинству простых смертных. К счастью, можно полностью игнорировать это обстоятельство и работать в своих программах с датами и временем точно так же, как вы это делаете на бумаге или в текстовом редакторе.

При этом нужно только помнить, что значения дат и времени — их буквальные значения — необходимо помещать в ограничивающую их пару знаков "решетки"'. Например, в следующем фрагменте программного кода сначала объявляются две переменные типа Date, а затем им присваиваются значения:

```
Dim dateWeddingDay As Date, dateTimeOfCeremony As Date
dateWeddingDay = #4/20/99#
dateTimeOfCeremony = #3:15:00 PM#
```

Как и в случае данных типа Currency, VBA автоматически отображает даты в формате, соответствующем местному стандарту. Например, выражение Format(#10/24/89#,"Long date") в США порождает строку "Tuesday, October 24, 1989", а в России — "24 Октябрь 1989 г.".

Вводдат

Печатать буквальные значения дат можно почти в любом виде, какой только возможен. Например, допустимо любое из следующих представлений:

```
#09/1/1998#
#Sep 25, 93#
#Janua 9 1905
```

Если редактор Visual Basic распознает введенные вами данные как допустимое значение даты, эти данные будут конвертированы в "краткое представление даты", заданное в соответствующей панели управления Windows. Если вы не укажете год, VBA добавит текущий год за вас. Такое конвертирование происходит сразу же, как только вы переместите текстовый курсор в другую строку, а не откладывается до момента, когда программа начнет выполняться.

Время имеет значение

Буквальные значения времени вводятся в формате *#часы:минуты:секунды символ#*, где *символ* — это AM или PM (означающие до полудня и после полудня соответственно). Например:

```
#10:45:00 PM#
#2:3:30 AM#
```

Печатать незначащие нули, как в #01:02:03 PM#, не обязательно, но VBA добавит их за вас, как только вы переместите курсор в другую строку программного кода. Точно так же вы можете опустить ненужные вам элементы в представлении времени, и VBA дополнит ваш ввод. Например, вы можете ввести только секунды, напечатав что-нибудь вроде #0:0:23#, а VBA изменит введенное значение на стандартное представление времени — в данном случае на #12:00:23 AM#.

Математика дат и времени

Сложение и вычитание дат в рамках обычных арифметических операций в VBA *возможны,* но, к сожалению, результаты такого сложения и вычитания не соответствуют обычным представлениям о датах. Например, #3/19/2005# - #3/19/2004# не дает в результате "1 год", как желательно было бы иметь, а дает #12/30/1900#. Такой результат обусловлен спецификой представления значений дат в VBA, но я не собираюсь здесь утомлять вас обсуждением этой специфики. Все, что вам нужно, — это знать о существовании в VBA двух функций, DateAdd() и DateDiff(), которые должны полностью удовлетворить ваши потребности в математике дат. Использование этих функций обсуждается в главе 11.

В противоположность датам, использование обычных арифметических операций со значениями времени дает *правильные* результаты. Взгляните, например, на выражения в следующем фрагменте программного кода:

```
Dim dateThen As Date
dateThen = #07:15 AM# + #12:00# ' = #07:15:00 PM#
dateThen = #07:15:00 AM# - #0:15 AM# ' = #07:00:00 AM#
dateThen = #07:15:15 AM# + #0:0:30 AM# ' = #07:15:45 AM#
```

В этом примере показан минимум того, что можно напечатать. Как всегда, VBA конвертирует введенные данные в полноформатное буквальное значение времени, например #0:0:30 AM# превратится в #12:00:30 AM#. И еще (как показано в представленном примере): если вы хотите получить правильные результаты при использовании арифметических операций со значениями времени, то используемые при этом переменные должны явно объявляться как переменные типа Date.

Информацию — в строку

Поскольку VBA с такой легкостью конвертирует данные одних типов в другие, строковые переменные оказываются нужными значительно реже, чем это может показаться на первый взгляд. Если единственной задачей является отображение нестрокового значения в виде, понятном обычному человеку, то для этого совсем не обязательно конвертировать такое значение в строковое. Лучше просто использовать данное числовое значение или дату — или даже значение типа Variant — в качестве аргумента некоторой функции или значения свойства объекта, для которых данный аргумент или значение должны быть строковыми.

В следующем примере используется сверхпопулярная функция MsgBox, отображающая данные строкового типа в небольшом диалоговом окне. В результате выполнения операторов

```
Dim dateЭтоДатаАНеСтрока As Date
dateЭтсДатаАНеСтрока = #17:23:16#
MsgBox dateЭтоДатаАНеСтрока
```

на экране отображается сообщение, подобное показанному на рис. 7.1.

Рис. 7.1. Показанный в окне сообщения текст не преобразовался в строку явно

А теперь взгляните на следующую ересь:

```
Dim intSmall As Integer, sngTall As Single
        Dim varYouAll As Variant
        intSmall = 3
        sngTall = 9.99E+33
        varYouAll = intSmall & sngTall
```

В результате выполнения этого фрагмента программного кода varYouAll будет содержать "39.99E+33". Присутствие в последнем операторе знака конкатенации (&) заставляет VBA конвертировать оба числовых значения в строковые, чтобы суметь выполнить эту конкатенацию.

Для чего строковые переменные *действительно* необходимы, так это для работы с нечисловыми символами, т.е. с буквами и знаками пунктуации. Их-то уж из числовых значений так просто не получить.

Из практики надежного программирования вытекает необходимость явного конвертирования числовых значений в строки, если требуется манипулировать ими как строками. Это уменьшает вероятность появления ошибок и делает программный код понятнее. И все же не мешает знать, что существует возможность вывода значений переменных, не требующая лишних усилий для их предварительной подготовки.

Кавычки в объявлениях строк

По той же причине не обязательно использовать кавычки, когда строковой переменной (т.е. переменной, объявленной как переменная типа String) присваивается дата, числовое или денежное значение. Как заботливая нянька, VBA конвертирует такие значения в строки безо всяких вопросов и нареканий. Так, в результате выполнения операторов

```
Dim strGSting As String
strGSting = #July 22, 1904#
```

переменная будет содержать строку "22.07.1904", пока какой-нибудь другой оператор не изменит ее. А теперь можете смеяться надо мной, но я рекомендовал бы использовать кавычки просто для того, чтобы быть уверенными, что ваша переменная содержит именно то, что вы собирались в нее поместить.

Функции VBA Asc и Chr конвертируют ANSI-символы в соответствующие им числовые коды и наоборот. С помощью функции Chr в строку можно поместить символы, которые в ней нельзя напечатать (например, кавычки). Функция Asc возвращает числовой код *первого* символа в строке.

Строки фиксированной длины

В VBA предусмотрено объявление строк двух типов. При стандартном объявлении строк (например, с помощью оператора Dim strMessage As String) создается переменная, которая может хранить строку любой длины. Вы можете изменить размер такой переменной, просто присвоив ей строку другой длины.

Но иногда удобнее зафиксировать длину строковой переменной. Это чаще всего нужно в тех случаях, когда приходится считывать и записывать данные файлов *прямого доступа,* что не предполагается рассматривать в этой книге, но обсуждается на соответствующей Web-странице (www.seldenhouse.com/vba). Пока что достаточно объяснений по поводу того, как объявлять строки фиксированной длины. Вот пример такого объявления:

```
Dim strFixed As String * 5
```

Объявленная таким образом переменная всегда будет содержать ровно 5 символов. Если ей присвоить более короткую строку, VBA добавит после символов этой строки нужное число пробелов. Если же переменной присвоить более длинную строку, VBA сохранит в переменной только 5 первых символов, отбросив остальные:

```
strFixed = "abc" ' теперь strFixed содержит "abc"
strFixed = "Fourscore and seven years" ' а теперь "Fours"
```

Глава 8

Управление потоком

В этой главе...

- **Использование управляющих структур для управления происходящим**
- **Проверка условий с помощью условных выражений**
- **Принятие решений с помощью операторов** `If...Then` **и** `Select Case`
- **Повторение выполнения действий с помощью операторов** `For...Next`, `For Each...Next` **и** `Do...Loop`
- Разветвления **с** помощью **операторов** `Go To`

Управляющие структуры — это операторы программного кода, определяющие, какие действия будут выполняться следующими, и использующие для этого некоторое условие, действительное во время выполнения программного кода. VBA предлагает полный ассортимент достаточно мощных управляющих структур. В этой главе мы обсудим каждую управляющую структуру VBA в отдельности **и** рассмотрим использование таких структур на практических примерах (ну, по крайней мере, на квазипрактических).

Укрощение диких программ с помощью управляющих структур

Управляющие структуры можно разбить на три главные группы — условные операторы, циклы и операторы With.

- ✓ *Условный оператор* определяет, какую из ветвей программного кода выполнять, в зависимости от того, какое значение (`True` или `False`) принимает некоторое условие. К условным операторам VBA относятся `If...Then` и `Select Case`.
- ✓ *Цикл* повторяет выполнение некоторого блока программного кода либо заданное число раз, либо до тех пор, пока некоторое условие не примет значение `True` или `False`. Если известно заранее, сколько раз необходимо выполнить цикл, то используйте `For...Next`, а если продолжение повторения программного кода зависит от выполнения некоторого условия, используйте `Do...Loop` (этот оператор доступен в нескольких вариантах). Чтобы повторить некоторые действия по отношению к объектам в коллекции, используйте цикл `For Each...Next`, который будет обсуждаться в главе 12.
- ✓ *Оператор* With позволяет выполнить множество действий с одним и тем же объектом без необходимости каждый раз указывать объект (см. главу 12).

Управляющие структуры привносят в программу ясность, организацию и... структурную стройность. Они позволяют с относительной простотой проследить тот путь, который программа может выбрать при ее выполнении.

Анатомия управляющих структур

Что делает управляющую структуру структурой, так это то, что управляющая структура является не просто отдельным оператором, а целым блоком операторов. Моделью для всех управляющих структур служит базовый оператор If ... Then:

```
If a < b Then ' Если а меньше Ь, то
     b = a ' положить значение Ь равным а,
     a = c ' а затем положить а равным с.
End If ' Это все - продолжить выполнение программы.
```

Каркас этой структуры составляют открывающий оператор, идентифицирующий структуру и задающий условие, и оператор, означающий конец структуры. Между этими двумя операторами находятся операторы, образующие тело структуры,— операторы, которые *выполняют* реальную работу.

Примерно такой общий вид имеют все управляющие структуры, правда, в некоторых из них условие, используемое структурой, помещается в последний оператор, а не в первый.

Вложенные управляющие структуры

Когда речь идет об управляющих структурах, *вложение* означает размещение одной структуры внутри другой, еще до оператора, означающего завершение первой структуры. VBA начинает работу со второй структурой, еще не закончив обрабатывать первую. Такое вложение оказывается необходимым при решении многих сложных реальных проблем. Управляющие структуры можно вкладывать одну в другую до любого уровня, какой вам кажется необходимым.

В следующем примере структура Do While... Loop вложена внутрь структуры If. . . Then, а еще одна структура If ... Then вложена в Do While... Loop:

```
If a < b Then ' начало внешней структуры If...Then
     Dc While b > c ' начало вложенного цикла Do While...Loop
          b = b - 1
          If c > d Then ' начало вложенной структуры If...Then
               d = a
          End If ' конец внутренней структуры If...Then
     Lcop ' конец цикла Do While...Loop
End If ' конец внешней структуры If...Then
```

Используйте отступы!

Правильное использование отступов служит основой создания понятного программного кода. В примере из предыдущего раздела каждая пара операторов, определяющих одну структуру, напечатана с одним и тем же отступом. При наличии отступов проще выяснить, например, какому из операторов If...Then соответствует данный оператор End If. Операторы, выполняемые в рамках данной структуры, тоже имеют один и тот же отступ, поэтому сразу видно, что все они находятся под юрисдикцией одной структуры.

Курс на использование условных выражений

Несмотря на простоту концепции, управляющие структуры можно отнести к наиболее мощным средствам программирования. С одной стороны, эти структуры "только" выбирают, какой из двух различных блоков программного кода следует выполнить. Но, с другой — когда в результате один блок программного кода выполнен, а другой — нет, вы имеете полное основание повторить за Фростом: "Это и явилось причиной всех различий".

Чтобы решить, выполнять ли некоторый блок программного кода, три следующие управляющие структуры VBA оценивают предложенное вами *условное выражение:* `Do . . . Loop`, `If. . .Then` и `Select Case`. Весь последующий материал этого раздела посвящен использованию условных выражений в этих структурах. Две другие структуры, `For. . . Next` и `For Each. . . Next`, условных выражений не используют.

Как работает условное выражение

Структуры `If. . .Then`, `Select Case` и `Do . . .Loop` принимают решение о последующих действиях на основе простого теста: какое значение принимает выражение — `True` (Истина) или `False` (Ложь)? Условием здесь может быть любое выражение VBA. (Не забывайте, что в VBA 0 эквивалентен `False`, а все другие значения рассматриваются как `True`.)

Чаще всего условные выражения строятся на основе какой-нибудь операции сравнения, применяемой к двум другим выражениям, входящим в данное. Набор операций сравнения, доступных для использования в VBA, достаточно подробно обсуждался в главе 7. Но основные идеи видны и из примеров операторов, помещенных в следующую таблицу. Каковы эти выражения — истинные или ложные? Это только управляющим структурам известно.

Выражение	Перевод на русский язык
`a < b`	a меньше b
`b = c`	b равно c
`colTBears("Генри") Is objCurrentBear`	Объект, хранящийся в коллекции `colTBears` под именем "Генри", является тем же самым объектом, что и объект, на который ссылается переменная `objCurrentBear`
`sqr(l/x * 29.3234) >= CDbl(strNumber) + 12`	Квадратный корень величины, равной 1, разделенной на x и умноженной на 29,3234, не меньше числового значения строковой переменной `strNumber` плюс 12

Условные выражения без операторов сравнения

Большинство условных выражений включают операции сравнения, но ничто не мешает создать условия, не использующие такие операции. Прежде чем привести примеры, не помешает подвести под них некоторую теоретическую базу.

Этот вопрос обсуждается здесь не для того, чтобы заставить вас создавать условные выражения, не использующие операций сравнения, а просто потому, что такие выражения создают многие программисты. Часто такие условные выражения присутствуют в текстах программ, публикуемых в компьютерных журналах и представленных в Internet. Но пока вы занимаетесь освоением VBA, я рекомендую вам включать операции сравнения в условные выражения хотя бы для ясности программного кода, даже если такие операции и не необходимы.

Теперь о деталях. Как вы знаете, значением логической переменной всегда является либо `True`, либо `False`. Поэтому выражение

```
boolДо18 ' это значение либо True, либо False
```

вполне подойдет в качестве условия в условном выражении, как, например, в случае

```
If boolДо18 Then
    ПредоставитьСкидку
End If
```

Но `True` и `False` имеют числовые значения, а поэтому в качестве условия можно использовать и любое числовое выражение. Все следующие выражения подходят для использования в качестве условий:

```
1234 ' всегда True
0 ' всегда False
True ' всегда True
False ' всегда False
intHowManyPets ' False, если intHowManyPets = 0
lngA + lngB + lngC ' False, если сумма = 0
```

Свойства объектов часто представляют логические значения, поэтому в качестве условий нередко используются выражения типа `frmHelpWindow.Enabled` и безо всяких операций сравнения.

Нельзя использовать строки и объекты в качестве условий сами по себе, хотя они вполне годятся для использования в качестве компонентов выражений, построенных на основе подходящей операции сравнения.

Использование логических операций в условиях

Логические операции (см. главу 7) сначала оценивают значения входящих в выражение двух выражений-компонентов как `True` или `False`, а затем, в соответствии с определенными правилами, на основе этих значений получается конечный результат — тоже `True` или `False`.

Самыми важными логическими операциями (точнее, теми из них, использование которых проще всего объяснить) будут `And`, `Or` и `Xor`. Следующая таблица объясняет, что эти операции выполняют.

Операция	Возвращает `True`	Примеры	Результат
And	Только если оба выражения принимают значения `True`	`3 * 2 = 6 And 12 > 11`	`True`
		`2 + 2 = 4 And 4 - 2 = 1`	`False`

Операция	Возвращает True	Примеры	Результат
Or	Если хотя бы одно из двух выражений принимает значение True	10 > 20 Or 20 > 10	True
		5 < 4 Or 6 < 5	False
Xor	Если только одно из двух выражений принимает значение True	5 + 5 < 9 Xor 5 + 5 = 10	True
		5 + 5 > 9 Xor 5 + 5 = 10	False

Если желаете, можете использовать две логические операции в одном условном выражении. Полюбуйтесь, например, таким выражением:

```
(a + b > 20 And c = 10) Or (objDoor.Open)
```

В переводе на русский язык этот фрагмент программного кода можно прочитать так: "Данное выражение есть истина, когда либо

a + Ь больше, чем 20, а с равно 10,

либо

свойство Open объекта obj Door принимает значение True".

Я советовал бы не перегружать одно выражение несколькими логическими операциями, разве что вы настолько сообразительны, что вам вообще не нужен компьютер. А чтобы гарантировать применение операций именно к тем выражениям, к которым нужно, используйте скобки. В предыдущем примере скобки вокруг выражения obj Door. Open гарантируют, что именно это выражение будет участвовать в операции Or первым.

Условные операторы If...Then

Условные операторы If. . .Then, а также их вариации If. . .Then. . .Else и If. . .ElseIf используются значительно чаще любых других операторов. Довольно часто операторы If. . .Then встречаются и в других главах, но в этой они уж точно в центре внимания.

Основная форма If...Then

Оператор If . . . Then выполняет некоторый блок программного кода, если условие, которое вы предложили этому оператору, принимает значение True, и не делает ничего, если условие принимает значение False. Синтаксис оператора следующий:

```
If условие Then
(операторы, выполняющиеся, когда условие = True)
End If
```

Простой пример использования оператора If. . .Then можно найти выше в разделе "Анатомия управляющих структур" этой главы.

Обратите внимание на следующие моменты, касающиеся операторов If. . .Then.

✓ Здесь условие должно быть условным выражением типа описанных выше в разделе "Курс на использование условных выражений". Если условие принимает значение True, то VBA выполнит операторы, размещенные между If. . .Then и End If.

- ✓ Обратите внимание, ключевое слово Then размещается в одной строке с If и выражением условие. При размещении Then в следующей строке (без указания переноса с помощью символа подчеркивания в первой) VBA генерирует ошибку.
- ✓ Не забывайте печатать завершающий оператор End If, При его отсутствии VBA не поймет, какой оператор должен быть последним в блоке.

Однострочные операторы If...Then

Если структура If...Then должна выполнить только один оператор, когда условие принимает значение True, всю эту структуру можно уместить в одну строку. В таком случае оператор End If не требуется — точнее, его быть не должно. Оператор

```
If curЦена > 20 Then MsgBox "Цена слишком высока!"
```

в результаге идентичен структуре

```
If сигЦена > 20 Then
    MsgBox "Цена слишком высока!"
End If
```

Операторы If...Then...Else

Если нужно, чтобы на основании одного условия программа выбирала между двумя альтернативными блоками программного кода, используйте оператор If...Then...Else. В том случае, если условие принимает значение True, то выполняется один блок программного кода, а если условие принимает значение False, то выполняется другой. Вот формальный синтаксис этого оператора:

```
If условие Then
(операторы, выполняющиеся, когда условие = True)
Else
(операторы, выполняющиеся, когда условие = False)
End If
```

Если условие имеет значение True, VBA выполняет первый блок операторов и, пропустив все остальные операторы структуры, переходит к строке программного кода, следующей сразу за оператором End If.А если условие имеет значение False,то выполняются только операторы, следующие за оператором Else.

В следующем примере выражение условие проверяет, является ли элемент управления в форме кнопкой. При положительном результате фон кнопки закрашивается красным цветом, все другие элементы управления — голубым:

```
If TypeOf ctlCurrentControl Is CommandButton Then
    ctlCurrentControl.BackColor = &HFF& ' красный
Else
    ctlCurrentControl.BackColor = &HFFFF00& ' голубой
End If
```

В этом примере предполагается, что переменная ctlCurrentControl уже содержит ссылку на соответствующий объект в форме. Ключевое слово TypeOf позволяет проверить, является ли объект, на который ссылается переменная, объектом нужного типа — в данном случае объектом CommandButton.

Использование операторов If...Then

Часто, прежде чем принять решение о действиях, которые должна выполнить программа, приходится проверять два или даже больше условий. Это — как в жизни. Когда вы пишете книгу по

VBA, то можете рассуждать примерно так: "Если я успею закончить книгу к установленному сроку и если не закончатся деньги, а также если курс рубля к доллару не изменится, то я смогу слетать в октябре на пару недель в Америку. Но если рубль упадет, то мне придется ехать в Тамбов[11].

В зависимости от ситуации, при этом потребуется добавить в структуру `If...Then` ключевые слова `ElseIf` или вложенные структуры `If...Then`.

Использование If...ElseIf

Используйте ключевое слово `ElseIf`, чтобы проверить дополнительное условие, если вы хотите выполнять определенные операторы только в случае, когда первое условие не принимает значение `True`. Ниже представлен синтаксис такой структуры:

```
If условие1 Then
(операторы, выполняющиеся, когда условие1 = True)
ElseIf условие2 Then
(операторы, выполняющиеся, когда условие1 = False,
            а условие2 = True)
ElseIf условие3 Then
(операторы, выполняющиеся, когда условие!
            и условие2 = False, а условие3 = True)
... (другие операторы ElseIf)
Else ' необязательное ключевое слово
(операторы, выполняющиеся, когда все условия = False)
End If
```

При этом ключевое слово `ElseIf` в структуре может повторяться любое число раз. Ключевое слово `Else` необязательно, но если оно присутствует, то должно быть в структуре последним,

Чтобы заставить работать этого монстра, представьте себе, что программа помогает вам вести учет в магазине по продаже фотоматериалов. Пленку, срок хранения которой истек, следует списывать и не учитывать в дальнейшем. Если же пленка еще не просрочена, ее нужно учесть в общем количестве пленок соответствующего типа.

Это делается в следующем фрагменте программного кода. Предполагается наличие объекта, представляющего ролик с пленкой, и наличие у этого объекта свойств для даты истечения срока хранения, типа пленки и признака цветности (как альтернативы черно-белого варианта).

```
If objRollOfFilm.ExpDate < Date Then
     MsgBox "Эта пленка не годится."
ElseIf objRollOfFilm.Type = "Слайдовая" Then
     intСлайдовые = intСлайдовые + 1
ElseIf objRollOfFilm.Color Then
     intЦветныеНегативные = intЦветныеНегативные + 1
Else
     intЧБНегативные = intЧБНегативные + 1
End If
```

Первый оператор сравнивает значение свойства `ExpDate` (срок хранения) с текущей датой. Если дата окончания срока хранения уже прошла, программа отображает окно сообщения с соответствующей информацией, и это все. Только если пленка все еще годится (т.е. если первое условие принимает значение `False`), будет выполнен первый из операторов `ElseIf`. Этот оператор проверяет свойство `Type` (тип) объекта, представляющего пленку. Если значением свойства является "`Слайдовая`", то следующий оператор увеличивает счетчик слайдовых пленок.

Если же пленка другого типа, программа переходит к следующему оператору `ElseIf`, в котором в качестве условия рассматривается свойство `Color` *(без* операций сравнения,

в предположении, что это свойство может принимать только значения True или False). Если значением свойства будет True, то пленка учитывается как цветная негативная, а если значение False, остается только перейти к выполнению Else — больше некуда — и учесть пленку как негативную черно-белую.

В структуре If. . .ElseIfвыполняются только операторы, ассоциированные с первым принимающим значение True условием. После выполнения этих операторов все оставшиеся ElseIf и Else пропускаются.

Вложениеоператоровif...Then

Вложенные операторы If. . .Then в некотором смысле являются противоположностями операторов If. . .ElseIf. Используйте эти операторы, когда для принятия решения нужно проверить дополнительное условие, но только если первое условие принимает значение True. Вложение одного оператора If...Then в другой подобно высказыванию: "Если X есть Истина и Y есть Истина, то я буду делать A, B и C".

Можно вкладывать один в другой операторы If. . .Then любых **видов** — If. . .Then. . .Else, If. . .ElseIfи разнообразные If...Then — и в любой комбинации. Вот схематическое представление пары вложенных операторов If...Then:

```
If условие1 Then
    If условие2 Then
    (операторы, выполняющиеся, когда условие1 и условие2 = True)
    ElseIf условие3 Then
    (операторы, выполняющиеся, когда условие1 и условие3 = True,
              а условие2 = False)
    (операторы, выполняющиеся, когда и условие1
              и условие2 = False, а условие3 = True)
    End If ' конец внутреннего блока If...Then
(другие операторы, выполняющиеся, когда условие1= True,
              независимо от того, чему равно условие2)
Else
(операторы, выполняющиеся только тогда, когда условие1= False)
End If
```

Следующий простой пример программного кода с использованием вложенных операторов If...Then отображает окно с поощрительным сообщением за высокие оценки и необходимый минимум часов классной работы:

```
If sngGPA > 3.5 Then
     If sngUnits > 10 Then
          MsgBox "Вы в поощрительном списке декана!"
     End If
End If
```

Использование логических операций в условиях

Использование логических операций в условных выражениях может быть более элегантной альтернативой использованию ElseIfи вложенных If... Then, когда нужно выполнить лишь одну ветвь пути, определяемого множеством условий.

Взгляните снова на последний фрагмент программного кода из предыдущего раздела. Ту же задачу можно выполнить с помощью лишь одного оператора If...Then, как здесь:

```
If sngGPA > 3.5 And sngUnits > 10 Then
    MsgBox "Вы в поощрительном списке декана!"
End If
```

Условные выражения с логическими операциями не помогут, когда необходимо использовать несколько ветвей пути решений. Попробуйте с помощью логических операций свести к примеру, подобному приведенному выше, следующий случай:

```
If sngGPA > 3.5 Then
    If sngUnits > 10 Then
        MsgBox "Вы в поощрительном списке декана!"
    Else
        MsgBox "Неплохо для прогульщика!"
    End If
End If
```

Мне на ум приходит только одно решение, все равно требующее двух операторов If...Then, хотя и не вложенных:

```
If sngGPA > 3.5 And sngUnits > 10 Then
        MsgBox "Вы в поощрительном списке декана!"
End If
If sngGPA > 3.5 And sngUnits <= 10 Then
        MsgBox "Неплохо для прогульщика!"
End If
```

использование операторов SelectCase

Операторы If...ElseIf и вложенные If...Then идеально подходят для принятия решений на основе проверки некоторого числа разных выражений. Если же приходится проверять *одно и то же* значение, сравнивая с различными выражениями, то самым подходящим обычно оказывается оператор Select Case. Синтаксис этого оператора следующий:

```
Select Case значение
    Case критерий1
        (операторы, выполняемые, когда значение
        удовлетворяет критерий1)
    Case критерий2
        (операторы, выполняемые, когда значение
        удовлетворяет критерий2)
... ' дополнительные операторы Case
    Case Else ' необязательный
        (операторы, выполняемые, когда значение
        не удовлетворяет ни одному из приведенных
        критериев)
End Select
```

Проверка условий в операторах Select Case

Структура Select Case не использует явным образом полные условные выражения, подобные тем, о которых шла речь выше (см. раздел "Курс на использование условных выражений"). Вы должны разбить каждое условие на две части, представленные как значение

и критерийN в предыдущем разделе при описании синтаксической конструкции этой структуры. Например, если у нас есть условное выражение

```
a + b > c
```

то значение можно определить как часть выражения, находящуюся слева от знака операции сравнения (a + b), а все, что останется, включая знак операции, — это критерийN(> c).

Пример оператора Select Case

Здесь явно не помешает пример, показывающий, как в действительности может выглядеть структура Select Case.

```
Select Case objRollOfFilm.Type
      Case "Слайдовая"
            intСлайдовые = intСлайдовые + 1
      Case "Цветная негативная"
            intЦветныеНегативные = intЦветныеНегативные + 1
      Case "ЧБ негативная"
            intЧБНегативные = intЧБНегативные + 1
      Case Else
            MgaBox "Неизвестный тип пленки."
End Select
```

В основном этот фрагмент программного кода делает то же, что и приведенный выше программный код для примера из раздела об операторах If. . .ElseIf (опущена только проверка срока хранения). Правда, с тех пор наш гипотетический объект, представляющий ролик пленки, похоже, был немного модифицирован — информация о цветности-бесцветности пленки теперь тоже представляется свойством Type, а не отдельным свойством Color, как раньше.

А если дело обстоит таким образом, то программе приходится работать только с одним значением — со значением, возвращаемым свойством Type, — но сравнивается это значение с несколькими из допустимых. Так что Select Case — это как раз то, что для нашего случая доктор прописал.

Первое применение оператора Case в данном примере эквивалентно применению If objRollOfFilm. Type = "Слайдовая" Then, т.е. если свойство Type объекта равно "Слайдовая", то программа выполняет следующий оператор, в противном случае она перейдет ко второму оператору Case.

Заметьте, что знака операции, присутствие которого кажется на первый взгляд логичным, в критериях нет. Причина в том, что в операторах Select Case равенство в качестве операции сравнения просто подразумевается.

Оператор Case Else

Если значение свойства Type не будет удовлетворять ни одному из критериев в операторах Case, то управление перейдет к оператору Case Else, который всегда находится в самом конце структуры Select Case. В предыдущем примере результатом этого будет сообщение об ошибке "Неизвестный тип пленки". Оператор Case Else необязателен, поскольку вполне вероятно, что вас устроит отсутствие всякой реакции на неудовлетворение всех критериев. Как и в данном примере, обычно нет никакой необходимости использовать оператор Case Else, разве что для того, чтобы программист проинформировал самого себя о не предусмотренных им значениях, хранящихся в программе.

Дополнительные сведения об операторе Case

В предыдущем примере критерии, заданные в операторах Case, были просто цветочками, детскими вопросами типа "Равно ли свойство Type тому-то и тому-то?" Но с помощью оператора Case вы можете задавать и значительно более сложные критерии.

Это проще всего показать на примере с числами. Предположим, ваш оператор Select Case открывается следующей строкой программного кода:

```
Select Case intВозрастПациента
```

В данном случае значением для сравнения будет переменная целого типа intВозрастПациента, представляющая возраст пациента клиники. В следующем списке представлено несколько типов сравнения, которые можно реализовать с помощью операторов Case, и приводятся соответствующие примеры.

- ✓ Можно сравнить тестовое значение с некоторым диапазоном значений, как в случае

```
Case 18 To 35
Messages("ВзрослыйМолодой").Print
```

- ✓ Обратите внимание на ключевое слово To, размещенное между значениями, определяющими границы диапазона. Сам диапазон включает и эти значения, и те, которые находятся между ними.

- ✓ Можно сравнивать значения и с помощью операций сравнения, отличных от операции =, как в случае

```
Case Is > 65
Messages("ВзрослыйПожилой").Print
```

- ✓ Здесь предполагается использовать перед операцией сравнения ключевое слово Is. На самом деле вы можете Is и не печатать — если вы его пропустите, VBA вставит это ключевое слово за вас.
- ✓ Можно в одном операторе Case указать несколько критериев для сравнения, как в случае

```
Case 0 To 5, 15, Is > 55
Messages("НапоминаниеОПрививке").Print
```

- ✓ Не забудьте разделить критерии для сравнения запятыми. Кстати, оператор Case с несколькими критериями эквивалентен выражению, построенному на основе использования операции Or, — если тестируемое значение удовлетворяет хотя бы *один* из критериев, следующие за Case операторы будут выполнены.

Повторение с помощью циклов

Управляющие структуры типа "цикл" используются тогда, когда необходимо повторить выполнение некоторого блока программного кода несколько раз. Повторение одного или нескольких операторов является главным средством выполнения многих математических вычислений, извлечения небольших порций данных из больших, применения одних и тех же действий по отношению к каждому из множества элементов некоторой группы.

VBA предлагает три типа структур, организующих циклы.

Тип цикла	Особенности работы
Do...Loop	Пока или до тех пор, пока некоторое условие имеет значение True
For...Next	Заданное число раз
For Each...Next	Для каждого объекта из коллекции объектов

При работе с вложенными циклами помните простое правило: внутренний цикл должен закончиться прежде, чем закончится внешний.

Циклы Do

Все возможные версии оператора Do ... Loop предназначены для повторения заданного блока программного кода неопределенно долго, пока не будет выполнено некоторое условие. Для того чтобы выяснить, продолжать цикл или нет, оператор Do ... Loop оценивает заданное условное выражение типа условных выражений, используемых для оператора If...Then и описанных выше в разделе "Курс на использование условных выражений".

Случаев, когда структуры Do...Loop полезны, просто не счесть. Вот несколько примеров.

- ✓ Отображение сообщения об ошибке снова и снова, пока пользователь не введет подходящую информацию в диалоговое окно.
- ✓ Чтение данных из файла на диске, пока не будет обнаружен конец файла.
- ✓ Поиск и подсчет числа вхождений некоторой строки символов в более длинной строке.
- ✓ Организация холостой работы программы в течение некоторого времени.
- ✓ Выполнение некоторых действий по отношению ко всем элементам массива.
- ✓ Выполнение (с использованием операторов If...Then) некоторых действий по отношению ко всем элементам массива или коллекции, удовлетворяющим определенным критериям (массивы и коллекции рассматриваются в главе 13).

Типы операторов Do...Loop

В VBA несколько видов циклов Do...Loop, но все они работают очень похоже.

Оператор	Выполняемые действия
Do...Loop	Повторяет выполнение блока программного кода, пока некоторый условный оператор внутри цикла не выполнит команду End Do
Do While...Loop	Начинает и повторяет выполнение блока программного кода, только если заданное условие принимает значение True
Do...Loop While	Выполняет блок программного кода один раз, а затем повторяет выполнение, пока заданное условие принимает значение True
Do Until...Loop	Начинает и повторяет выполнение блока программного кода, только если заданное условие принимает значение False
Do...Loop Until	Выполняет блок программного кода один раз, а затем повторяет выполнение, пока заданное условие принимает значение False

ОператорDoWhile...Loop

Оператор Do While... Loop можно считать прототипом структуры Do. Вот его синтаксис:

```
Do While условие
 (операторы, выполняющиеся, когда условие = True)
Loop
```

Обнаружив оператор Do While, VBA начинает проверять *условие*. Если представляющее условие выражение оказывается равным False, VBA игнорирует остальную часть цикла и переходит сразу к операторам программы, следующим за оператором Loop. Если же *условие* принимает значение True, VBA выполняет операторы в блоке. Достигнув оператора Loop, VBA возвращается опять к оператору Do While, чтобы проверить условие снова.

Как правило, один или несколько операторов в теле цикла могут изменить значение выражения, представляющего условие, поэтому *условие* может стать равным False. В этом случае VBA прерывает выполнение цикла и, пропустив все его операторы, переходит к выполнению операторов, следующих за оператором Loop. Но если условие все еще равно True, операторы цикла будут выполнены еще раз.

Весь процесс повторяется до тех пор, пока в некоторый момент *условие* не станет равным False. Другими словами, нет ограничений на число повторений выполнения блока программного кода в цикле. В предположении, что подача энергии не прекратится и компьютер не отключается, цикл будет продолжаться вечно, если *условие* никогда не примет значение False.

ПримериспользованияDoWhile...Loop(точнее,двапримера)

Следующий пример, основанный на использовании двух операторов Do While... Loop, переставляет в обратном порядке цифры в числе, предложенном пользователем. Наверное, этот пример не имеет никакой практической ценности, но он эффектный, работает в любом VBA-приложении и достаточно компактный, чтобы иллюстрировать работу циклов Do:

```
Sub ReverseTheDigits()
Dim intOriginalNumber As Integer
Dim intOneDigit As Integer, strBackwardsNumber As String
Do While intOriginalNumber < 10
     intOriginalNumber =
          InputBox("Напечатайте число, большее 9.")
Loop
Do While intOriginalNumber
     intOneDigit = intOriginalNumber Mod 10
     strBackwardsNumber = strBackwardsNumber & intOneDigit
     intOriginalNumber = Int(intOriginalNumber / 10)
Loop
MsgBox strBackwardsNumber
End Sub
```

Объяснение*примера*

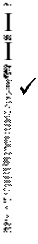
В первом цикле Do проверяется, что значение, введенное пользователем, неотрицательно и что во введенном числе не меньше двух знаков (а иначе, что же в нем переставлять?). Когда программа выполняет этот цикл впервые, значением переменной intOriginalNumber будет 0, поскольку до этого переменной не присваивалось никакого значения. Нуль меньше 10, поэтому условие принимает значение True, и VBA начинает выполнение цикла.

Цикл содержит один оператор, отображающий окно ввода, в котором пользователя просят ввести подходящее число. После того как число введено, оператор Loop отсылает про-

цесс выполнения к началу цикла, где введенное число проверяется. Цикл завершит свою работу только тогда, когда проверка покажет, что было введено подходящее число. (Обратите внимание на то, что в данном примере опущены некоторые важные пункты проверки правильности вводимых данных, наподобие проверки того, является ли число целым и не является ли больше максимально допустимого для переменных типа Integer.)

После подтверждения правильности введенного значения VBA переходит к следующему циклу. Условием выполнения этого цикла является положительное значение, содержащееся в переменной. Поскольку ненулевые числа порождают True, можно вместо Do While intOriginalNumber > 0 писать просто Do While intOriginalNumber — оба варианта работают одинаково.

В самом цикле простая процедура в три строки разбирает цифры исходного числа, начиная справа налево, и использует их в обратном порядке для построения новой строки. Чтобы понять, как работает цикл, не обязательно разбираться в программном коде внутри цикла, но, я думаю, следующие пояснения вам не помешают.



- ✓ В первой строке используется операция Mod, чтобы присвоить переменной intOneDigit остаток от деления числа на 10. Поскольку число делится на 10, остатком будет однозначное число, представленное последней (справа) цифрой исходного числа.
- ✓ Во втором операторе цифра, полученная в первой строке, добавляется в конец создающейся новой строковой величины.
- ✓ В третьем операторе число снова делится на 10, но на этот раз результат сохраняется в исходной переменной. При таком делении просто отбрасывается правый знак числа. Заметим, однако, что перед тем, как результат присваивается переменной, он обрабатывается функцией Int. Использование функции Int здесь необходимо, поскольку в противном случае результат был бы округлен и это изменило бы знаки числа по сравнению с введенными изначально.
- ✓ В процессе прохода цикла значение intOriginalNumber уменьшается и наконец становится равным 0 после того, как все цифры будут обработаны (любая цифра задает число, меньшее 10, а функция Int отбрасывает дробную часть числа). В стране VBA нуль означает False, поэтому цикл завершится, и программа покажет число с обратным расположением цифр.

Другие операторы Do

Разобравшись в базовой форме оператора Do While...Loop, несложно понять и его вариации В этом разделе рассмотрены три варианта циклов Do, четвертый обсуждается ниже, в разделе "Когда использовать Do без While или Until".

Do..Loop While

Разница между операторами Do While...Loop и Do...Loop While очень простая: Do While... Loop имеет условие в самом начале цикла, а в Do...Loop While условие идет в самом конце.

В структуре Do While... Loop вход в цикл происходит вообще, только если условие принимает значение True, когда программа впервые обращается к циклу. Если сначала условие принимает значение False, цикл не выполнится ни разу.

В структуре Do...Loop While, напротив, цикл всегда выполняется по крайней мере один раз, поскольку сначала программа проходит весь блок программного кода, а лишь затем выполняется проверка условия, чтобы повторить выполнение операторов цикла, если условие окажется равным True. Используйте структуру Do ... Loop While тогда, когда значение условия для проверки устанавливается в блоке программного кода цикла.

Другим случаем, когда подходящим решением оказывается структура Do... Loop While, является обработка объекта (типа строки или массива), в котором больше одного элемента. Если вы точно знаете, что элементов не меньше одного, можно допустить выполнение операторов цикла один раз, а затем повторять выполнение столько раз, сколько нужно для оставшихся компонентов.

Do Until...Loop

Операторы Do While...Loop и Do Until...Loop функционально эквивалентны, т.е. с помощью любого из них будут выполняться одни и те же операторы, если используемые в них условные выражения представляют противоположные условия. Подобным образом дополняют один другой и операторы Do While...Loop и Do Until...Loop.

Представьте себе, что вы ведете передачу по кулинарии и рассказываете своим слушателям рецепт. Вы можете сказать: "Продолжайте вымешивать масло, пока в нем заметны неоднородности" или "Продолжайте вымешивать масло до тех пор, пока оно не станет совершенно однородным". И первое, и второе означают одно и то же.

Подобным образом взаимно дополняются операторы Do While и Do Until. Если в операторе Do While условием является A = B, то эквивалентным ему оператором будет Do Until с условием A o B.

Do или не Do

Цикл Do в своем условном выражении обеспечивает всю необходимую информацию, на основании которой VBA может принимать решение о продолжении или завершении выполнения цикла. К сожалению, в реальности дело не всегда обстоит так идеально, и иногда изменение другого условия внутри цикла требует немедленного выхода из цикла, для чего в VBA предусмотрен оператор Exit Do.

Действующий только внутри цикла Do, оператор Exit Do немедленно прекращает выполнение цикла и передает управление оператору, следующему сразу после цикла. В следующем примере выполняется конкатенация строковой переменной с некоторой имеющейся строкой, но если переменная содержит больше одного символа, цикл прекращается:


```
Do While strA <= "2"
      If Len(strA) > 1 Then
            Exit Do
      End If
      strB = strB & strA
      strA = GetNextCharacter
Loop
```

Обычно необходимость в операторе Exit Do может возникнуть только во вложенных в цикл операторах If...Then и Select Case. В этих случаях цикл выполняется нормально до тех пор, пока не происходит что-то особое или не появляется непредусмотренное значение. Оператор Exit Do можно использовать также и для отладки, для временной отмены выполнения цикла, чтобы не размещать апостроф (') в начале строк целого блока программного кода.

КогдаиспользоватьDobezWhileилиUntil

В стандартных операторах Do While/Until... Loop условие можно проверять либо в начале, либо в конце цикла. Но что делать, если вы хотите проверить некоторое условие *внутри* цикла?

В этом случае используйте оператор Do...Loop без While и Until. Но при этом вам понадобится вложить в цикл либо оператор If, либо Select Case. Вложенный условный оператор может содержать оператор Exit Do, что позволит программе прекратить выполнение цикла, когда заданное условие окажется выполненным.

Вот схематическое представление оператора Do ... Loop:

```
Do
(операторы, выполняющиеся при каждом проходе цикла)
     If условие Then
          Exit Do
     End If
(операторы, выполняющиеся, только если цикл продолжает выполняться)
Loop
```

Как видите, эта техника годится в том случае, когда нужно выполнить некоторые операторы в цикле, независимо от выполнения или невыполнения условия, а также если выполнение цикла необходимо прервать в зависимости от некоторого другого условия.

Например, часто приходится проверять правильность введенных пользователем данных относительно целого ряда критериев. В следующем примере с помощью структуры Do...Loop цикл ввода повторяется до тех пор, пока пользователь не введет правильную букву ответа:


```
Sub GetAnAnswer()
Dim strAnswer As String
   strAnswer = InputBox("Enter your answer (A-E)")
Do
   If strAnswer = "" Then
   strAnswer = InputBox("You didn't enter anything." _
   & " Please type a letter between A and E.")
   ElseIf Len(strAnswer) > 1 Then
   strAnswer = InputBox("Your answer should be" _
   &  " only one letter long. Please try again.")
   Elself strAnswer < "A" Or strAnswer > "E" Then
   strAnswer = InputBox("You typed an invalid" _
   & " character. Type a letter from A to E.")
   Else
   Exit Do
   End If
Loop
End Sub
```

Эта программа выполнит оператор Exit Do только тогда, когда все три условия, заданные в операторах If и ElseIf,будут выполнены.

В качестве альтернативы для достижения той же цели можно предложить использовать Do While True в первой строке структуры, задающей цикл. Поскольку True (константа) всегда имеет значение True, условие всегда выполняется, и, следовательно, VBA обязательно начнет и будет продолжать выполнять цикл. Поэтому, для того чтобы завершить выполнение такого цикла, внутри него должен присутствовать оператор Exit Do.

Повторение под управлением циклов For...Next

Если уже перед выполнением цикла известно, сколько раз он должен выполняться, используйте цикл For... Next. Число проходов цикла задается значениями начало и конец, которые могут быть целыми числами, переменными и даже сложными выражениями. В процессе выполнения цикла переменная счетчик хранит информацию о числе выполненных проходов цикла. Когда значение счетчика становится равным значению конец, выполнение цикла завершается.

Упрощенно синтаксис структуры For... Next можно представить так;

```
For счетчик = начало To конец
(операторы, выполняющиеся при каждом проходе цикла)
Next счетчик
```

В следующем примере для отображения сообщения при каждом проходе цикла в процедуре используется окно Immediate (в редакторе Visual Basic окно Immediate открывается нажатием клавиш <Ctrl+G>):

```
Sub CountToTen()
Dim j As Integer
    For j = 1 To 10
        Debug.Print "Дубль № " & j
    Next j
End Sub
```

В этом примере начало и конец являются буквальными числовыми значениями. Когда цикл начинается, j получает значение 1; другими словами, переменной счетчик присваивается значение начало. После каждого прохода цикла оператор Next j увеличивает значение j (на 1) и отсылает управление снова в начало цикла. Когда j достигает значения 10, выполнение цикла прекращается.

Важные замечания по поводу циклов For... Next

Старайтесь, чтобы ваш программный код всегда оставался понятным. Используйте 1 в качестве начального значения для цикла For... Next, если только у вас нет серьезных причин выбрать для этого другое число.

Такие серьезные причины на самом деле бывают, например, когда значение счетчика используется в самом цикле (но, уточню, не меняется в цикле). Если в цикле предпринимаются некоторые действия по отношению к последовательно пронумерованным элементам типа номеров страниц, то в качестве значений для величин начало и конец можно использовать реальные значения номеров этих элементов. В частности, при работе с массивами обычно начало выбирается равным 0 (подробности — в следующем разделе).

В операторе Next счетчик, завершающем цикл For... Next, имя переменной-счетчика указывать не обязательно — ключевое слово Next автоматически вычислит следующее значение счетчика и отошлет VBA в начало структуры. Однако определенно стоит приучить себя включать имя счетчика в оператор Next. Тогда даже в случае нескольких вложенных циклов For...Next вы всегда

сможете с первого взгляда сказать, какому циклу принадлежит данный оператор Next. Не меняйте значение счетчика внутри цикла For. . .Next. Поскольку счетчик является обычной переменной, можно — а иногда и заманчиво — создать программный код, меняющий значение счетчика. Не поддавайтесь!

Стоит ошибиться со счетчиком и можно пропустить пару важных шагов или заставить цикл выполняться бесконечно.

Хотя и можно значение конец для цикла For . . . Next задать переменной, изменение значения такой переменной после начала выполнения цикла не повлияет на процесс выполнения. Цикл все равно будет выполнен столько раз, сколько задано оригинальным значением конец.

Циклы For...Next и массивы

Циклы For. . .Next особенно полезны при работе *с массивами,* именованными хранилищами для множеств элементов данных. Подробно массивы обсуждаются в главе 13, но дискуссия о циклах For. . . Next осталась бы неполной без нескольких слов о пользе таких циклов в совокупности с этими имеющими большое практическое значение корзинами для данных.

С помощью цикла For. . .Next несложно заполнить массив множеством вычисленных значений, как в следующем примере:

```
Dim intМассивКвадратов (14) As Integer
For a = 0 To 14
     intМассивКвадратов(a) = a * a
Next a
```

Этот пример программного кода начинается с объявления массива из 15 целых значений (15, а не 14, поскольку обычно VBA приписывает первому элементу' в массиве индекс 0). Затем используется цикл For . . . Next, в котором каждому элементу массива, от 0 до 14, присваивается значение. Обратите внимание на то, что переменная a используется не только как счетчик, но и как *индекс* массива, указывающий на номер элемента в массиве.

Вложенные циклы For...Next

Подобно другим управляющим структурам VBA, циклы For. . .Next можно вкладывать один в другой — или в другие управляющие структуры, —- так глубоко, как это необходимо. Следующий совершенно бесполезный фрагмент программного кода иллюстрирует эту возможность:

```
Dim sngC  ' наверное, C означает "случайное число"
Randomize ' инициализация генератора случайных чисел
For A = 1 To 5
     sngC = Rnd()
     For B = 1 To 5
          Debug.Print sngC * Rnd()
     Next B
Next A
```

Если вы достаточно наблюдательны, то без труда сможете проследить за шагами, которые делает VBA в процессе выполнения этого программного кода.

1. Предварительная подготовка.

   Программный код начинается с объявления переменной и инициализации генератора случайных чисел VBA.

2. **Начало внешнего цикла For. . .Next.**
   VBA вызывает функцию Rnd, чтобы присвоить переменной sngC случайное значение.
3. **Начало внутреннего цикла For. . .Next.**
   Этот цикл вычисляет пять других чисел, повторяя вызов функции Rnd при каждом проходе цикла. Результат отображается в окне **Immediate.**
4. **Завершение внутреннего цикла после выполнения им всех пяти вычислений.**
   Теперь снова продолжится выполнение внешнего цикла. Подчиняясь оператору Next A, VBA возвращается в начало внешнего цикла.
5. **Пп. 2 и 3 повторяются еще для четырех проходов внешнего цикла.**

Этот пример, вероятно, тривиален, но его можно заставить выполнять очень полезную работу, если немного подправить. Допустим, вам нужно создать мультимедиа-программу, случайным образом выбирающую пять компакт-дисков и проигрывающую по пять случайным образом выбранных фрагментов с каждого из дисков. В предположении, что вы знаете, как программировать выбор компакт-дисков и воспроизведение записей, предыдущий пример будет как раз подходящей заготовкой.

Использовать вложенные циклы For. . .Next очень удобно при обработке многомерных массивов, если организовать циклы так, чтобы каждый из них соответствовал одному измерению массива.

Немедленный выход с помощью Exit For

Оператор Exit For предназначен для немедленного прекращения выполнения цикла до того, как программа доберется до его конца. Обычно этот оператор используется в условном выражении (If. . . Then или Select Case), вложенном в главный цикл For. . .Next.

Удобно использовать Exit For, когда проверяются массивы, есть ли в них недопустимые значения, когда неправильные данные приводят к прекращению выполнения процесса.

Скажем, вы узнали о том, что какой-то злой гений ввел неправильную информацию в ваш прайс-лист и при этом оставлял свой "фирменный" знак. Поэтому при внесении изменений в цены из-за инфляции вам придется проверить для каждого из массивов, не испорчены ли в нем данные. Вот пример программного кода, который делает всю эту работу:

```
For p = 1 To varДлинаМассива
     If varЦена(p) = "Здесь был Ваня!" Then
          MsgBox "Данные испорчены."
          Exit For
     End If
     varЦена(p) = varЦена(p) * sngCOLA
Next p
```

Задание шага цикла

Полный синтаксис оператора For. . .Next включает необязательное ключевое слово Step (шаг) в первой строке структуры, как, например, в следующем фрагменте программного кода:

```
Sub ListOddNumbers()
Dim strOddNumbers As String
For F = 1 To 33 Step 2
     strOddNumbers = strOddNumbers & F & " "
```

```
Next F
MsgBox "Нечетными числами между 1 и 33 являются:" & _
     Chr(13) & strOddNumbers
End Sub
```

В этом примере цикл создает строку, содержащую все нечетные числа из диапазона, определяемого значениями начало и конец цикла. Значение переменной-счетчика при каждом проходе цикла увеличивается на 2 — шаг, заданный значением аргумента Step. Поскольку начальным значением задано 1, F всегда будет нечетным, что упрощает задачу программирования операторов внутри цикла.

В общем, аргумент Step сообщает VBA, как вычислять следующее значение счетчика по достижении конца цикла. Обычно, когда аргумент Step явно не указан, счетчик увеличивается на 1. Указав аргумент Step, вы можете задать другое значение, на которое нужно будет увеличивать счетчик. При этом можно задать и отрицательное значение, тогда счетчик будет уменьшаться.

И еще раз повторяю — не будьте слишком хитроумными. Необычные значения аргумента Step могут порождать ошибки, которые трудно обнаружить просто потому, что ошибки при этом тоже бывают необычными. Поэтому используйте явное задание аргумента Step только тогда, когда этого явно требует ситуация, да и в этих случаях лучше сначала попытаться найти другие решения, уменьшающие вероятность появления осложнений.

Управление потоком с помощью Go To

Если ваша программа вдруг стала неуправляемой, укажите ей, куда идти, — с помощью передачи управления другому участку программного кода в процедуре. Оператор Go To в совокупности со специальным оператором метки в месте назначения позволяет по желанию перемещаться от одной точки в процедуре к другой. *Метка* — это оператор, просто отмечающий некоторое место в программном коде. Чтобы задать метку, напечатайте ее имя (согласно правилам создания имен в VBA), а после него — двоеточие.

Пример использования Go To

В следующем примере оператор Go To направляет поток программы из главной части процедуры к метке SpecialValue, если встречается необычное значение:

```
Function GoToExample (ItemNumber As Integer)
     Dim intR As Integer
     Select Case ItemNumber
          Case 2412
               Go To SpecialValue
          Case Is < CutOffValue
               DoSomething
          Case >= CutOffValue
               DoHardlyAnything
     End Select
     (операторы, выполняющие какие-то действия)
     GoToExample = intR
     Exit Function
SpecialValue:
```

```
    DoSomethingSpecial
    GoToExample = -intR
End Function
```

Обратите внимание на формат метки `SpecialValue` — она размещается в своей отдельной строке и заканчивается двоеточием. Двоеточие здесь просто обязательно — при его отсутствии VBA непременно расстроится и выведет сообщение об ошибке.

Оправдания Go To

Использование оператора `Go To` считается в программировании признаком низкого качества. Дело в том, что использование этого оператора превращает программный код в "спагетти", когда путь выполнения программы скачет туда-сюда через всю программу. "Спагетти" трудно распутывать, а программный код по мере увеличения в нем числа операторов `Go To` быстро становится невозможным для чтения. Всегда, когда это возможно, используйте управляющие структуры, позволяющие программе выполняться последовательно.

Но иногда оператор `Go To` оказывается самым естественным способом для того, чтобы заставить программу делать то, что вам нужно. Может быть, ваша голова уже до предела нагружена невероятным сплетением вложенных циклов и условных операторов, требующих сложного множества критериев. В таких случаях оператор `Go To` проложит прямую дорогу к выходу из лабиринта, только не используйте его слишком часто.

Глава 9

"Бронированный" программный код: отладка и устранение ошибок

В этой главе...

- Каталог всевозможных ошибок, стремящихся "одурачить" вашу программу
- Принципы борьбы с ошибками
- Замечательная возможность — режим паузы
- Все средства отладки редактора Visual Basic, включая команды Step и окна Immediate, Locals и Watch
- Использование редактора Visual Basic как калькулятора с помощью окон Immediate и Watch
- Изощренные приемы отладки, уменьшающие риск появления ошибок при выполнении программы

Создание программного кода в VBA — это только полдела. Заставить этот программный код работать и работать так, как нужно, — вот что требует больше всего усилий. Вылавливание и уничтожение ошибок становятся решающим моментом в создании программы, и нам с вами пришло время выяснить, как это делается в VBA. В этой главе мы обсудим также приемы программной обработки ошибок, позволяющие программе грациозно выходить из трудных ситуаций, которые случаются во время ее выполнения.

See возможные неприятности происходят обязательно

Программа, создаваемая с помощью VBA (как и любого другого языка программирования), обычно сопровождается тремя следующими типами ошибок.

- ✓ **Ошибки компиляции.** Синтаксические и другие ошибки, сразу же делающие невозможным выполнение программы.
- ✓ **Логические ошибки.** Изъяны проектирования программы, в результате которых программа делает то, что вы *не* планировали, или *не* делает того, что вы планировали.
- ✓ **Ошибки выполнения.** Приводят к остановке выполнения программы вследствие либо логических ошибок, либо возникновения ситуации, не предусмотренной в программе (подробно об этом — ниже, в подразделе "Откуда берутся ошибки выполнения").

Изо всех трех типов ошибок синтаксические, определенно, самые простые и для обнаружения, и исправления. Поэтому я коснусь синтаксических ошибок только вкратце, чтобы основное внимание уделить приемам выявления и искоренения ошибок двух других типов; по сути, они и являются настоящими ошибками.

Исправление синтаксических ошибок

Если вы допустите синтаксическую ошибку, редактор Visual Basic сообщит вам об этом почти сразу же, не дожидаясь, когда вы дадите указание выполнить программу. Если вы напечатаете что-то такое, что редактор Visual Basic не поймет, символы в строке с непонятным программным кодом станут красными, а как только точка ввода перейдет из этой строки на другую, вы получите сообщение, подтверждающее наличие ошибки и с некоторым разъяснением ее (при условии, что на вкладке Editor (Редактор) диалогового окна Options (Параметры) отмечен флажок Auto Syntax Check (Автоматическая проверка синтаксиса); чтобы открыть это диалоговое окно, выберите Tools⇨Options из меню). Например, если вы напечатаете If x = 3 и забудете напечатать Then, появится сообщение Compile error: Expected: Then or GoTo (Ошибка компиляции: ожидалось Then или GoTo).

Некоторые синтаксические ошибки VBA не замечает до тех пор, пока программа не начнет выполняться. Например, VBA не будет поначалу возражать, если вы укажете в операторе GoTo ссылку на несуществующую метку или вызовете несуществующую процедуру— по этому поводу вы получите сообщение об ошибке компиляции уже после начала выполнения программы.

Когда известно, где допущена синтаксическая ошибка, исправить ее уже несложно. Если вы сомневаетесь в правильном написании имени, объявления или вызова процедуры или функции, имени переменной, оператора создания нового экземпляра объекта или использования управляющей структуры, обратитесь к соответствующим главам этой книги или подходящим разделам справки по VBA.

Энтомология для программистов

После того как вы выловите и уничтожите все ошибки компиляции, а ваша программа начнет выполняться, никто не осудит вас за внезапно нахлынувшее чувство гордости и удовлетворения. Но если программа вдруг сообщит, что 2 + 2 = 22, или если весь текст в документе окажется розовым, то знайте — где-то в программном коде все же затаилась ошибка.

Причиной подобных ошибок являются ошибки в логике программы. При выполнении программа всегда делает то, что ей приказали делать вы, —- проблема заключается в том, что с помощью созданного вами программного кода вы *заставили* программу делать не то, что в действительности *хотели* заставить ее делать. Вы и должны отыскать те операторы, которые привели к неправильным действиям. Процесс отыскания ошибок называется *отладкой,* и в этом разделе обсуждаются все средства и приемы, которые помогут вам в вашей отладочной кампании.

Ясно, что приемы, которые используются для выявления и исправления логических ошибок, годятся и для предотвращения появления ошибок выполнения, вытекающих из логических. Но поскольку ни одна программа не в состоянии предвидеть абсолютно все возможные ситуации, в любой программе, предназначенной для использования другими людьми, необходимо предусмотреть программный код для обработки вероятных ошибок выполнения.

Тест, тест, тест

Хорошо, когда ошибки сами заявляют о себе, например, с помощью явно неправильных ответов, изменения цветов экрана на очень странные или появления сообщений об ошибках. К сожалению, часто логические ошибки оказываются не такими "крикливыми", а многие ошибки выполнения вообще случаются только иногда, при определенном стечении обстоятельств. Пожалуй, не стоит предполагать, что ваша программа будет работать правильно и надежно сра-

зу же после того, как "'вылупится", гораздо лучше запланировать достаточно времени на ее тщательное тестирование в самых разных условиях.

Вот несколько советов по тестированию программ.

- ✓ Если профамма предназначена для работы с разными типами документов, то и попробуйте ее в работе с разными типами документов. Попытайтесь работать с программой, когда в VBA-приложении открыто несколько документов и когда — ни одного.
- ✓ Посмотрите, как выполняется программа, когда окно документа находится в разных состояниях (развернуто, свернуто или восстановлено).
- ✓ Попробуйте выполнить программу, когда в окне документа выделены различные элементы или группы элементов.
- ✓ Если программа требует ввода информации от пользователя в специальном окне или в специальной форме, опробуйте все вероятные и невероятные типы ввода и посмотрите на результат. Например, если вводимое значение предполагается целым, попробуйте ввести значение с плавающей запятой, дату и строку символов, чтобы узнать, как отреагирует на это программа.
- ✓ Если программа работает со значениями дат и времени, попробуйте задать ей самые разные даты (в том числе и 29 февраля) и самое разное время суток (в том числе и полночь). Испробуйте не только разные значения даты и времени для переменных, но и проверьте работу самой программы при условии разных дат и разного времени. Для этого не нужно вставать среди ночи или ждать следующего года— просто установите для своего компьютера подходящие показания системных времени и даты.

Во время тестового запуска программы внимательно наблюдайте за ходом ее выполнения, чтобы не пропустить даже малейших признаков ошибок. Внимательно проанализируйте документ и убедитесь в наличии всех изменений, которые должны были появиться после выполнения программы,— непредусмотренных изменений тоже быть не должно. Обращайте внимание на точность и правильный формат каждого из результатов, выводимых профаммой на экран или помещаемых в документ. В идеале, нужно проследить и за значениями всех переменных в ходе выполнения программы, чтобы быть уверенным, что при этом тоже не возникает никаких неожиданностей. (О том, как это сделать, я расскажу в следующем разделе.) И ясно, что если уж VBA сообщит вам об ошибке выполнения, то наличие проблемы очевидно.

Если вы обнаружите ошибки, к отладке нужно отнестись со всей серьезностью. В противном случае лучше вообще считать, что ошибок нет, потому что их не бывает и быть не может.

Комбинации клавиш для отладки

В табл. 9.1 перечислены комбинации клавиш, которые используются при отладке. Подробно каждую из них я рассмотрю дальше в настоящей главе.

Таблица 9.1. Отладка

Действие	Комбинация клавиш
Построчное выполнение кода (пошаговое)	<F8>
Построчное выполнение кода без построчного выполнения отдельных процедур	<Shift+F8>
Выполнение кода до точки вставки	<Ctrl+F8>

Действие	Комбинация клавиш
Добавление точки останова	<F9>
Удаление всех точек разрыва	<Ctrl+Shift+F9>
Определение следующей инструкции, которая будет выполняться	<Ctrl+F9>
Добавление контролируемого выражения	<Shift+F9>
Выполнение отладочного кода или возвращение сведений об ошибке обращения к процедуре	<Alt+F5>
Пошаговое выполнение отладочного кода или возвращение сведений об ошибке обращения к процедуре	<Alt+F8>

Сделайте паузу!

Ключ к отладке программы кроется в режиме паузы VBA. *Режим паузы* — это временная остановка выполнения программы на некотором операторе в программном коде. Поскольку в этом случае программа еще "живет", вы получаете возможность проверить текущие значения всех ее переменных. Кроме того, начиная с этого момента, вы получаете возможность использовать команды Step, чтобы продолжить выполнение программы в пошаговом режиме, по одному оператору за шаг, и наблюдать за соответствующими изменениями значений переменных,— при этом будет видно, получают переменные ожидаемые значения или нет. Подробности такой работы с переменными и командами Step будут рассматриваться в этой главе позже.

На рис. 9.1 показан вид окна редактирования VBA-процедуры в режиме паузы. Если не считать желтой подсветки строки и стрелки на поле слева, указывающей на оператор, который должен выполняться следующим, то это окно редактора Visual Basic выглядит практически так же, как и при обычном редактировании программного кода.

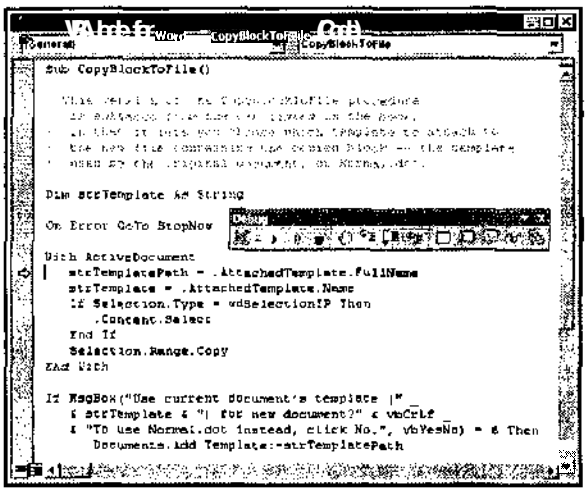


Рис. 9.1. Процедура VBA в режиме паузы

На самом деле, в режиме паузы вы можете редактировать свой программный код прямо во время выполнения программы, внося изменения и добавляя совершенно новые строки программного кода по необходимости или по своему желанию. Это предусмотрено не для забавы — это очень удобное средство отладки, преимуществами которого нужно обязательно научиться пользоваться (подробнее об этом — ниже, в разделе "Добавление и редактирование программного кода в режиме паузы").

Вход в режим паузы

Имеется целый ряд возможностей для перевода программы в режим паузы, который можно сравнить со стоп-кадром анимации.

- ✓ Запуск программы сразу в режиме паузы с помощью команды Step Into (см. ниже раздел "Сквозь программу по шагам").
- ✓ Назначение *точки останова* строке программного кода. После того как программа при выполнении дойдет до оператора, с которым связана точка останова, выполнение прервется, и программа перейдет в режим паузы.
- ✓ Помещение в программный код оператора Stop. После выполнения этого оператора программа переходит в режим паузы, приготовившись продолжить выполнение с оператора, следующего за оператором Stop.
- ✓ Щелчок на кнопке Break (Пауза), выбор Run⇨Break из меню или нажатие комбинации клавиш <Ctrl+Break> во время выполнения программы. Используйте эту возможность для восстановления контроля над программой, которая по каким-то причинам не желает останавливаться сама. Никто не сможет предугадать, где при этом программа остановится, но зато после остановки у вас появится возможность увидеть, где это произошло.
- ✓ Создание контролируемых переменных типа Break When Value Is True (стоп при значении Истина) или Break When Value Changes (стоп при изменении значения). Тогда программа перейдет в режим паузы, как только значение контролируемой переменной станет равным True или как только ее значение изменится.

По-другому программа может перейти в режим паузы, когда возникает ошибка выполнения. VBA отображает диалоговое окно с описанием возникшей ошибки (рис. 9.2). В этом окне щелчок на кнопке End (Закончить) останавливает выполнение программы совсем, а щелчок на кнопке Debug (Отладка) переводит выполнение программы в режим паузы. На вкладке General (Общие) диалогового окна Tools⇨Options вам предоставляются некоторые возможности для выбора круга тех ошибок, появление которых должно останавливать выполнение программы.

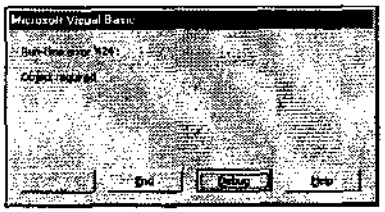


Рис. 9.2. Диалоговое окно, сообщающее об ошибке выполнения

Размещение точек останова в программе

Если есть подозрение, что ошибка содержится в каком-то конкретном сегменте программного кода, поместите точку останова как раз перед этим сегментом. На рис. 9.3 видно, как редактор Visual Basic представляет точку останова на экране — в виде довольно жирной точки на полях в окне редактирования.

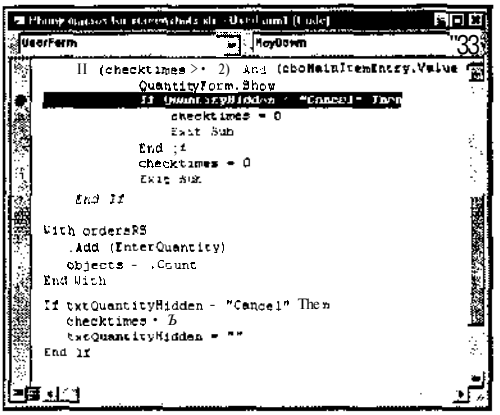


Рис. 9.3. При назначении точки останова редактор Visual Basic подсвечивает строку программного кода, перед которой должно остановиться выполнение программы

После того как точка останова размещена, вы имеете возможность выполнить программу на полной скорости вплоть до этой точки, пропустив отладку той части программного кода, которая (как вы надеетесь) не содержит ошибок. Когда VBA доберется до оператора, которому назначена точка останова, выполнение программы приостановится. В режиме паузы вы сможете проверить значения переменных и с помощью команд Step проследить за тем, как эти значения меняются при последовательном выполнении каждого из операторов в подозрительном блоке.

Чтобы разместить точку останова, просто щелкните на полях окна редактирования слева от соответствующей строки программного кода. Можно разместить точку останова и с помощью клавиатуры — для этого поместите курсор в нужную строку и нажмите <F9>.

Можно разместить столько точек останова в разных строках программы, сколько вы сочтете необходимым. Нельзя только размещать точки останова в строках комментариев и строках с операторами, которые VBA на самом деле не выполняет, например в строках с объявлениями переменных.

Помните о том, что VBA останавливает программу и переключается в режим паузы после выполнения оператора, непосредственно предшествующего оператору с точкой останова. Другими словами, оператор с точкой останова останется пока невыполненным— с него предполагается продолжить выполнение программы после выхода из режима паузы.

Удаление точек останова

Исправив ошибки в программном коде или отказавшись на время от попыток их исправления, вы захотите удалить ненужные точки останова. Удаление точек останова даст возможность VBA при следующем запуске выполнять программу без лишних задержек. Удаляется

существующая точка останова точно так же, как и ставится — щелчком на полях в окне редактирования или нажатием клавиши <F9>.

Чтобы удалить все назначенные точки останова сразу, выберите из меню Debug⇨Clear All Breakpoints или нажмите <Ctrl+Shift+F9>. Точки останова моментально уйдут в историю — никакая команда Undo (Отменить) не поможет вам восстановить их.

Выяснение места остановки

Когда программа находится в режиме паузы, редактор Visual Basic подсвечивает оператор, который предполагается выполнить следующим. Для гарантии на полях около строки с этим оператором в окне редактирования программного кода появляется стрелка. То, что вы увидите, показано на рис. 9.4 (но чтобы представить его в цвете, пожалуй, без фантазии не обойтись).

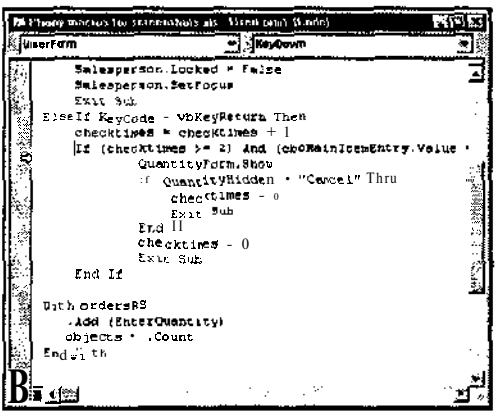


Рис. 9.4. Не заметить строку с оператором, который должен будет выполняться следующим, просто невозможно (тем более, что эта строка обязательно будет на экране)

Если строка с оператором, выполняющимся следующим, присутствует на экране, не заметить ее подсветку просто невозможно. Но что делать после того, как вы полистаете текст программы или переключитесь в окно другого модуля? В таком случае найти этот оператор без специальных средств будет уже сложнее.

Здесь поможет специально предусмотренная для такого случая команда Show Next Statement (Показать следующий оператор). Выберите Debug⇨Show Next Statement, и редактор Visual Basic быстренько доставит вас к нужному оператору. Есть также команда, позволяющая выбрать в качестве следующего другой оператор, но она будет обсуждаться позже, в подразделе "Назначение другого следующего оператора".

Альтернатива точкам останова: оператор Stop

Точки останова использовать очень просто, но они имеют один недостаток: они временны. Если отлаживаемая программа достаточно сложна, скорее всего, вам не удастся привести ее в полный порядок за один раз. Точки останова не сохраняются в программном коде, поэтому, открыв проект в редакторе Visual Basic в следующий раз, вы их уже не увидите.

Эта проблема решается с помощью оператора Stop. Поместите этот оператор в программный код. и в соответствующем месте программа перейдет в режим паузы. Вместе с другими операторами программы операторы Stop будут сохраняться при сохранении проекта.

Желаете пример, господа? Извольте:

```
...
intДанныеСМарса = ПолучениеДанныхСМарса(1.5454)
Stop
MsgBox "Результат " & intДанныеСМарса /Z
...
```

Взглянув на этот пример, вы можете догадаться, что здесь оператор Stop используется для того, чтобы прозондировать подозрительные результаты, отображаемые в окне сообщения. Выполнение оператора Stop заставит перейти программу в режим паузы сразу после того, как процедура-функция ПолучениеДанныхСМарса присвоит свое значение переменной. Вы получите возможность проверить значения обеих переменных, используемых для вычислений в следующей за оператором Stop строке, чтобы выяснить, какая из переменных стала источником проблемы.

Выход из режима паузы

Когда вы достаточно насладитесь режимом паузы (по крайней мере на этот раз), дайте указание VBA продолжить нормальное выполнение программы. Для этого используются команды, находящиеся на тех же местах, что и команды для первоначального запуска программы. В режиме паузы соответствующие пункт меню Run (Выполнить) и кнопка панели инструментов со своих мест никуда не денутся, но изменят свои названия — они будут теперь называться Continue (Продолжить). По-прежнему клавишным эквивалентом для вызова соответствующей команды будет <F5>. Продолжая выполнение, программа может снова перейти в режим паузы, если доберется до следующей точки останова или возникнут иные условия, при которых активизируется режим паузы.

Чтобы совсем прекратить выполнение программы, используйте команду Reset (Остановить выполнение). Для этого можно использовать кнопку Reset или пункт Reset меню Run. Нажатие <Alt+F4> останавливает выполнение вообще любой программы в Windows, в том числе и VBA-программ. Но прежде чем нажать <Alt+F4>, убедитесь, что вы выполняете свою VBA-программу, потому что в противном случае закроется либо редактор Visual Basic, либо ваше VBA-приложение, либо какая-то другая программа Windows, оказавшаяся активной в тот момент.

Сквозь программу по шагам

Как и все хорошие отладчики, редактор Visual Basic дает возможность выполнять программу по шагам, по одному оператору за шаг. Такое замедление — просто находка для тех случаев, когда появляется подозрение на наличие логических ошибок, имеющих привычку прятаться в укромных местах программ.

Чтобы использовать все преимущества *пошагового выполнения* программы, желательно видеть, как в процессе выполнения программы изменяются значения переменных. В этом всегда готово помочь чудесное средство редактора Visual Basic, называемое Auto Data Tips (Автоматические подсказки значений), но еще более подробная информация предоставляется в окнах Locals (Окно локальных переменных) и Watches (Окно контролируемых выражений), которые к тому же позволяют при необходимости изменить значения представленных в них элементов. Все эти возможности будут обсуждаться ниже.

Начнем же, пожалуй, с трех команд Step редактора Visual Basic, а именно: Step Into, Step Over и Step Out. Все три доступны либо через меню **Debug,** либо с помощью кнопок панели инструментов **Debug,** либо следующих комбинаций клавиш.

Команда	Комбинация клавиш
Step Into	<F8>
Step Over	<Shift+F8>
Step Out	<Ctrl+Shift+F8>

Вход в процедуры

Команда Step Into (Войти в процедуру) используется, если нужно выполнить процедуру по шагам в порядке естественного выполнения операторов. Каждый раз, когда применяется эта команда, VBA выполняет следующий оператор программы и снова переходит в режим паузы, чтобы вы имели возможность увидеть произошедшие при этом изменения. Удобнее всего вызывать команду Step Into нажатием клавиши <F8>.

Свое название эта команда получила из-за того, что при ее использовании происходит вход в любую из вызываемых в программе процедур: если следующим оператором оказывается вызов процедуры типа Sub или Function, то применение команды Step Into приведет к открытию вызванной процедуры в окне редактирования, чтобы вы могли пройти эту процедуру по шагам и увидеть все, что происходит.

Команда Step Into доступна не только в режиме паузы. Если нужно выполнить программу по шагам с самого начала, запустите ее с помощью команды Step Into. В таком случае команда Step Into начинает выполнение с той процедуры, в которой оказывается текстовый курсор, причем процедура всегда выполняется с самой первой строки.

Обход процедур и выход из них

Команда Step Over (Обойти процедуру) вызывается клавишами <Shift+F8> и работает подобно команде Step Into, но со следующими двумя отличиями:

- ✓ при вызове процедуры команда Step Over не приводит к пошаговому выполнению вызываемой процедуры;
- ✓ с помощью команды Step Over нельзя начать выполнение программы в режиме паузы.

Если оператор, который должен выполняться следующим, вызывает некоторую процедуру, команда Step Over выполнит эту процедуру без перерывов, целиком, сделав сразу все имеющиеся в процедуре глупости и не дав вам проверить промежуточные результаты. Обход процедур очень экономит время, когда вы уверены, что вызываемая процедура работает так, как нужно.

Команда Step Out (Выйти из процедуры) вызывается клавишами <Ctrl+Shift+F8> и прекрасно дополняет команду Step Over. После входа в процедуру в режиме ее пошагового выполнения вы можете решить, что все в порядке или что ошибка найдена и исправлена. Или, намереваясь щелкнуть на кнопке **Step Over,** вы можете случайно щелкнуть на кнопке Step Into (лично у меня такое происходит регулярно).

В любом из таких случаев нет необходимости проходить по шагам оставшуюся часть процедуры. Чтобы пройти остаток с полной скоростью, используйте команду Step Out. VBA быстро доставит вас в ту процедуру, из которой вы пришли, подсветив оператор, следующий за оператором с вызовом процедуры.

Основные приемы отладки

В режиме паузы поток выполнения программы не рассматривается как нечто "замороженное" только потому, что программа выполняется. VBA оказывается достаточно сообразительным, чтобы позволить вносить коррективы прямо по ходу дела. В частности, в режиме паузы можно редактировать программный код и изменять порядок, в котором должны выполняться операторы.

Добавление и редактирование программного кода в режиме паузы

Все возможности окна редактирования программного кода остаются в силе и в режиме паузы. Вы имеете возможность печатать новые операторы и редактировать или удалять уже имеющиеся. И главное, в большинстве своем эти изменения сработают незамедлительно, став частью выполняемой программы. Так, вы можете объявлять переменные, чтобы сразу же использовать их в вычислениях в комбинации с уже существующими переменными программы. Но есть также изменения (например, изменение типа переменной в ее объявлении), которые заставят VBA прекратить выполнение программы.

Назначение другого следующего оператора

Представьте, что вы, двигаясь сквозь свою программу в пошаговом режиме, вдруг обнаруживаете, что в следующем операторе содержится большая ошибка. Вместо того чтобы выполнить этот оператор и тем самым ввести свою программу в "штопор", вы можете обойти этот оператор и не выполнять его вообще, пока когда-нибудь в будущем не исправите ошибку.

Редактор Visual Basic почти всегда позволяет с помощью клавиш'либо мыши назначить другой оператор для выполнения следующим. При этом вы можете перемещаться в программном коде как вперед, так и назад. В дополнение к возможности пропустить неправильный программный код, вы можете повторять выполнение некоторого фрагмента программного кода до тех пор, пока не поймете, как он работает.

Только знайте, что переменные при этом будут хранить те значения, которые они имели перед началом внесения вами изменений. Эти значения могут сильно отличаться от тех, которые переменные имели бы в соответствующей точке при естественном ходе выполнения программы. Если необходимо, назначьте переменным значения по своему усмотрению с помощью окон Immediate, Watches или Locals, которые будут обсуждаться в этой главе ниже.

Вот как назначить другой следующий оператор.

- ✓ С помощью клавиатуры переместите точку ввода в строку с нужным вам оператором, нажмите <Ctrl+F9> или выберите Debug⇨Set Next Statement из меню.
- ✓ С помощью мыши перетащите желтую стрелку на полях в окне редактирования программного кода к строке с новым следующим оператором (рис. 9.5).

Еще один способ пропустить программный код

Есть еще один способ пропустить часть программного кода, на который падает подозрение в содержании ошибки. Этот метод годится и для режима паузы, и для обычного режима редактирования перед запуском программы на выполнение. Как вы уже знаете, можно напечатать апострофы в начале каждой строки того фрагмента программного кода, который нужно пропустить. Тем самым строки программного кода превратятся в строки комментария. Единственный недостаток этого способа- на такое "комментирование" с последующим "раскомментированием" нужно время. Правда, можно "комментировать" сразу целый блок с помощью одного щелчка на кнопке Comment Block (Превратить блок в комментарий) панели инструментов Edit (Правка) (см. главу 6).

Но часто для пропуска фрагмента программного кода самым удобным оказывается назначение метки и временное использование ссылающегося на нее оператора GoTo. Например, при выполнении следующего фрагмента программного кода VBA пропустит все операторы между GoTo ПослеПропуска и оператором, следующим за меткой ПослеПропуска:

```
...
GoTo ПослеПропуска
    A = B + C
    D = A + E / F
    G = B + D
ПослеПропуска:
    MsgBox '" & Format(Now, "dddd"
...
```

Когда пропущенный фрагмент понадобится снова, просто удалите строку с оператором GoTo или превратите ее в комментарий.

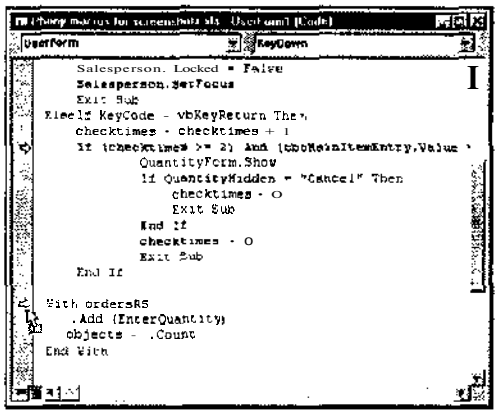


Рис. 9.5. Назначение другого оператора, предназначенного для выполнения следующим, дает возможность повторять выполнение фрагментов программы, чтобы разобраться в них

Автоматические подсказки

Замечательная возможность автоматической подсказки значений в VBA позволяет видеть текущее значение любой переменной в программе. Находясь в режиме паузы, задержите на секунду указатель мыши на имени нужной вам переменной, и на экране появится окно подсказки — небольшой прямоугольник с именем и текущим значением переменной в нем (рис. 9.6).

Если же прямоугольники с подсказками значений на экране не появляются, проверьте, чтобы был установлен флажок Auto Data Tips (Автоматическая подсказка значений) на вкладке Editor (Редактор) диалогового окна Options (Параметры).

Редактор Visual Basic позволяет также увидеть тип и область видимости переменной, хотя соответствующая информация и не появляется автоматически. Поместите курсор между буквами или около имени переменной и нажмите <Ctrl+I>, и на экране появится окно подсказки Quick Info (Краткая справка) с этой информацией. Для этого не обязательно находиться

в режиме паузы, что особенно удобно при работе с большими программами, когда объявления переменных оказываются где-то далеко в самом начале программы. Пример того, что вы можете увидеть в прямоугольнике краткой справки, показан на рис. 9.7.

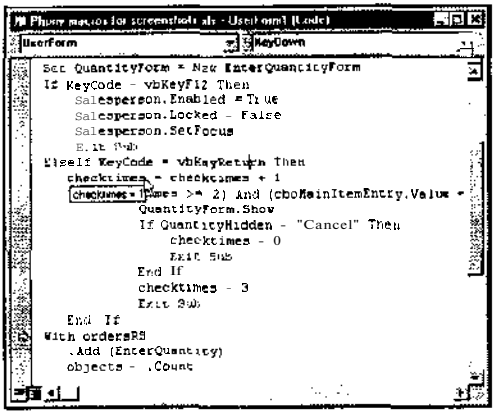


Рис. 9.6. В маленьком прямоугольнике подсказки рядом с указателем мыши отображается значение соответствующей переменной

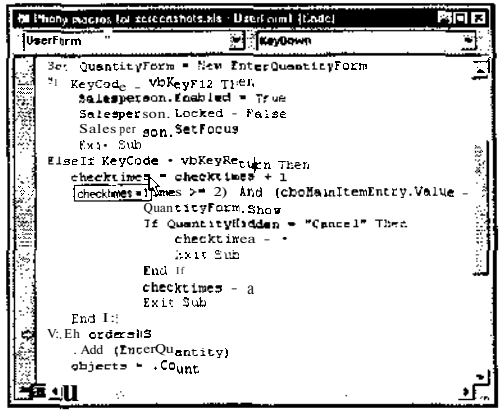


Рис. 9.7. Отображение подсказки Quick Info для переменной

Немедленное вознаграждение в окне Immediate

Чтобы открыть окно Immediate (Окно немедленного выполнения команд, рис. 9.8), нажмите <Ctrl+G> или выберите View⇨Immediate Window из меню. Окно Immediate предоставляет следующие возможности:

- ✓ видеть результаты вычислений и значения переменных, вывод которых можно направить в это окно с помощью метода Debug.Print;
- ✓ выполнять отдельные операторы сами по себе, не помещая их в процедуры: чтобы выполнить оператор в окне Immediate, просто напечатайте его там и нажмите <Enter>.

"Зачем это нужно?" — спросите вы. Отвечаю.

- ✓ Окно Immediate можно использовать просто как калькулятор. Напечатайте в нем выражение типа

```
Print (27 * 398) + 1414
```

затем нажмите <Enter>, и вы *немедленно* получите результат (см. рис. 14.8). При работе в окне Immediate вам нет необходимости указывать объект Debug.

- ✓ Можно направить в окно Immediate вывод промежуточных значений переменных и выражений при выполнении программы, разместив в подходящих строках программы операторы, вызывающие метод Debug.Print. По завершении выполнения программы вы получите возможность просмотреть сразу все выведенные значения (рис. 9.9), чтобы выяснить, все ли они правильны, а не рассматривать их по отдельности в разное время при пошаговом выполнении программы.

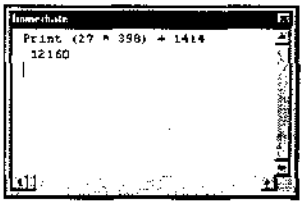


Рис. 9.8. Окно Immediate в действии

✓ В режиме паузы в окне Immediate можно отобразить значение любой переменной или свойства объекта с помощью оператора Print или изменить значение с помощью стандартного оператора присваивания. Можно также стандартным способом вызывать процедуры. При этом вы должны понимать, что в режиме паузы выполняемые в окне Immediate операторы могут иметь доступ только к переменным, объектам и процедурам, находящимся в области видимости той процедуры, которая выполняется в данный момент. Другими словами, результат выполнения оператора в окне Immediate будет тем же, что и результат печатания этого оператора в выполняемой в данный момент процедуре и работы его в ней.

Интересный факт. В окно Immediate можно перетащить текст, выделенный в окне редактирования программного кода, и тогда вам не придется печатать длинные выражения или имена переменных снова (но знайте, если при перетаскивании вы не будете удерживать нажатой клавишу <Ctr!>, фрагмент программного кода не скопируется в окно Immediate, а переместится туда из окна редактирования). Клавиша <F1> в окне Immediate работает точно так же, как и в окне редактирования программного кода, вызывая справку для ключевого слова, в котором находится текстовый курсор. А вот автоматические подсказки значений здесь не действуют.

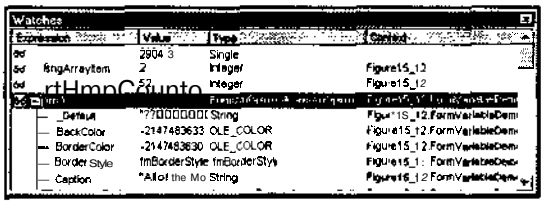


Рис. 9.9. Пример вывода программы, использующей метод Debug. Print

Все переменные под присмотром в окне Locals

Если у вас на экране еще есть место, где можно разместить окно Locals (Окно локальных переменных), оно обязательно должно присутствовать на экране все время, пока вы занимаетесь отладкой программы в режиме паузы. Отобразить окно Locals на экране можно щелчком на соответствующей этому окну кнопке панели инструментов Debug или с помощью View ⇨ Locals Window из меню.

Как видно из рис. 9.10, в окне Locals автоматически отображаются имена, значения и типы данных всех переменных, доступных в выполняемой в данный момент процедуре. Эта информация обновляется редактором Visual Basic после выполнения любого из операторов, так что вы всегда будете видеть в окне Locals текущие значения переменных.

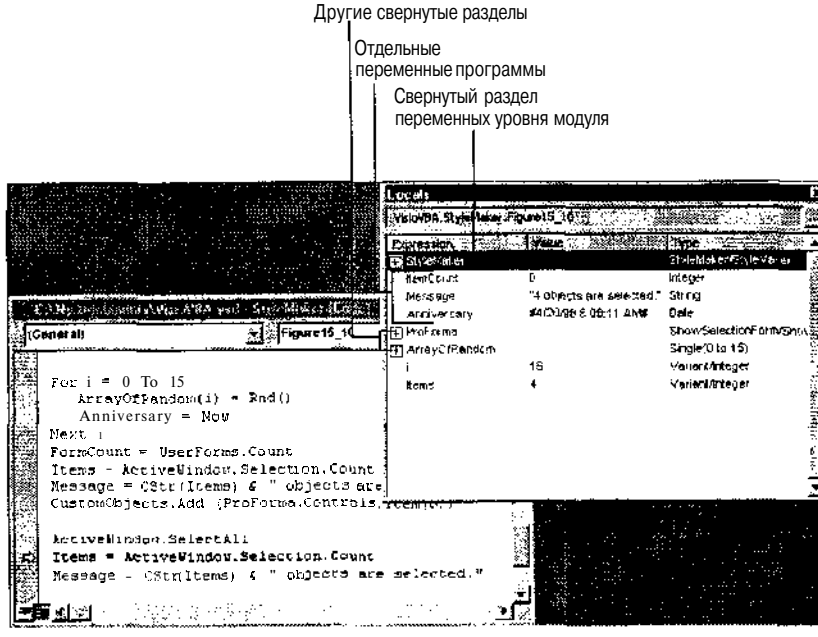


Рис. 9.10. Вид окна Locals в режиме паузы

Механика процесса

Как и ряд других окон редактора Visual Basic, окно Locals по умолчанию закреплено, но может отображаться и в виде свободно перемещаемого окна. Основы использования окон редактора Visual Basic обсуждались в главе 5.

Если нужная вам информация не помещается в столбце, помните о том, что ширину столбцов можно менять. Поместите указатель мыши на разделитель между заголовками столбцов и, когда он превратится в двунаправленную стрелку, перетащите разделитель влево или вправо, чтобы установить нужную ширину столбца.

Если снова взглянуть на рис. 9.10, вы увидите, что некоторые строки в окне Locals сообщают информацию об обычных переменных, а некоторые — о массивах, переменных пользовательских типов и объектах. Такие элементы не имеют собственных значений, а содержат другие переменные или элементы иных типов. Изначально после запуска процедуры в окне Locals они представлены в виде свернутых разделов, так что вы не увидите содержащихся в них подчиненных элементов. Чтобы развернуть любой из свернутых разделов, щелкните в прямоугольнике с плюсом слева от имени раздела (то, что можно увидеть, развернув раздел, показано на рис. 9.11).

В развернутых уже на самом верхнем уровне иерархии окна Locals будут показаны только переменные, явно или неявно объявленные в самой выполняемой в данный момент процедуре. Чтобы увидеть переменные уровня модуля, доступные всем процедурам данного модуля, разверните самый первый из разделов в окне Locals. Этот раздел всегда принадлежит модулю, содержащему выполняемую процедуру. В окне Locals не появляются открытые переменные или переменные из других проектов.

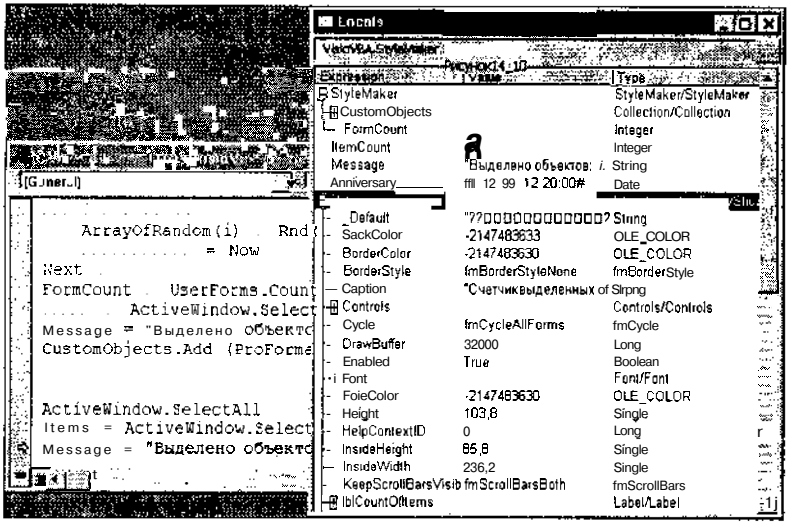


Рис. 9.1 /. Видокна Locals для той же процедуры, что и на рис. 9.10, но разделы, там свернутые, теперь развернуты

Зачем редактировать значения переменных

Прежде чем сообщить вам, как менять значения переменных, я думаю, нужно сказать пару слов о том, зачем их менять, Вот несколько случаев, когда это оказывается удобным.

- ✓ Предыдущая строка программного кода содержит ошибку, в результате которой переменной присвоено неправильное значение. Вы это заметили, но хотели бы продолжить выполнение программы. Прежде чем продолжить, неплохо было бы исправить значение переменной, чтобы ошибка не повлияла на последующие операторы.
- ✓ Вы хотели бы проверить выполнение программы при разных значениях переменной, не меняя текста самой программы. В таком случае используйте команду Set Next Statement (Назначить оператор следующим), чтобы повторить выполнение фрагмента программы несколько раз, задавая при этом разные значения в окне Locals.
- ✓ Реальные значения, которые программа будет получать в процессе ее использования, ей сейчас недоступны (например, программа должна считывать информацию из базы данных или из Internet). В этом случае с помощью окна Locals вы можете имитировать получение программой подходящих значений.

Как редактировать значения переменных

Для того чтобы изменить значение переменной в окне Locals, выполните следующее.

1. **Щелкните два раза (но не делайте двойной щелчок) на текущем значении переменной так, чтобы оказалось подсвеченным только это значение.**
   После щелчка в любом другом месте строки выделенной окажется вся строка.
2. **Напечатайте новое значение.**
   Если значение должно быть строковым, заключите его в кавычки, а если это значение даты, поместите его между парой знаков #.
3. **Если вы передумали менять значение переменной, нажмите <Esc>, а чтобы подтвердить изменения — <Enter>.**

Если редактор Visual Basic сочтет предложенное вами значение недопустимым, вы получите сообщение об ошибке с некоторыми объяснениями по поводу того, что именно оказалось неправильным.

Можно менять значения любых переменных, в том числе значения элементов массивов и элементов переменных пользовательских типов. Только, чтобы получить доступ к этим элементам, не забудьте развернуть раздел, соответствующий нужному массиву или переменной пользовательского типа. В случае объектной переменной у вас нет возможности изменить ссылку на объект, но зато можно менять свойства соответствующего объекта (за исключением тех, которые предназначены только для чтения). И опять же, можно увидеть редактируемые элементы, только развернув раздел, соответствующий объектной переменной (см. рис. 14.11).

Ключевое средство отладки: окно Watches

Когда вы освоите работу с окном Locals, работа с окном Watches (Окно контролируемых выражений) покажется просто забавой. Окно Watches делает, по сути, то же самое, что и окно Locals, но с одним очевидным отличием — те выражения, значения которых будут показаны в этом окне, должны выбрать *вы*. Примерный вид окна Watches показан на рис. 9.12.

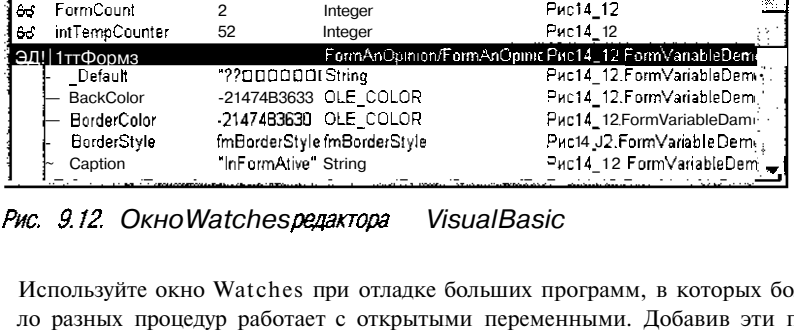
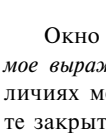


Рис. 9.12. Окно Watches редактора Visual Basic



Используйте окно Watches при отладке больших программ, в которых большое число разных процедур работает с открытыми переменными. Добавив эти переменные в окно Watches, вы сможете контролировать их значения, независимо от того, какая из частей программы выполняется в данный момент.

Окно Watches появится на экране автоматически, как только вы зададите *контролируемое выражение*. (О том, что такое контролируемое выражение, вы узнаете во врезке "Об отличиях между окнами Watches и Locals".) Если на экране не хватает места, вы всегда можете закрыть это окно, щелкнув на кнопке закрытия в его верхнем правом углу.

Об отличиях между окнами Watches и Locals

Кроме того, что появляющиеся в окне Watches объекты выбираете вы, окно Watches отличается от окна Locals следующим.

- ✓ Каждая из строк окна Watches позволяет контролировать значение любого допустимого в VBA выражения, а не только отдельной переменной. Например, вполне годятся выражения типа `(X - Y) > 15, "Выбранный цвет - " & strColor` или даже просто `2 + 2`. Именно поэтому в случае окна Watches говорится о контролируемых выражениях.

✓ Из-за того что в окне Watches можно контролировать переменные из любого модуля или процедуры проекта, в этом окне добавлен еще один столбец Context (Контекст). В этом столбце указывается область видимости, в рамках которой отображается значение данной переменной или выражения. Если выполняемая процедура находится вне заданного контекста, вы и увидите только пометку <Out of context> (Вне контекста).

Добавление контролируемых выражений

Если вы любите свободу выбора, вам обязательно понравится иметь дело с окном Watches. Приготовьтесь удивиться тому, сколько есть разных способов добавления контролируемых выражений в это окно.

Правда, независимо от выбранного в конце концов способа, начать нужно с окна редактирования и выделить в нем переменную или выражение, значения которых требуется контролировать. Здесь "выделить" означает подсветить все выражение, как в любом текстовом редакторе, пользуясь мышью или клавишами со стрелками.

Если нужно контролировать значение отдельной переменной (другими словами, элемента, не являющегося свойством объекта или частью пользовательского типа данных), то выбирать весь элемент целиком не нужно — достаточно поместить текстовый курсор внутри имени этой переменной.

Перечислим способы, которыми можно добавить контролируемые выражения.

- ✓ Щелчок на выделенном тексте правой кнопкой мыши и выбор из появившегося контекстного меню пункта Add Watch, открывающего диалоговое окно Add Watch.
- ✓ Выбор Debug⇨Add Watch из меню, в результате чего также открывается диалоговое окно Add Watch.
- ✓ Щелчок на кнопке Quick Watch панели инструментов Debug или выбор Debug⇨Quick Watch из меню. Редактор Visual Basic отобразит на экране подтверждающее сообщение с описанием почти уже добавленного вами контролируемого выражения, но у вас не будет возможности каким-либо образом это выражение изменить.
- ✓ Перетаскивание выделенного выражения в окно Watch. При этом контролируемое выражение создается без каких-либо дополнительных промежуточных шагов.

А если вы предпочитаете делать все своими руками, то можете в окне редактирования программного кода не выбирать ничего, а сразу щелкнуть на кнопке Add Watch и просто напечатать интересующее вас выражение вручную в соответствующем поле диалогового окна Add Watch.

Работа с окном Add Watch

На рис. 9.13 изображено окно Add Watch (Добавление контролируемого выражения), возникающее на экране в результате выбора Debug⇨Add Watch из меню или выбора пункта Add Watch из контекстного (вызываемого щелчком правой кнопки мыши) меню в окне редактирования программного кода. В окне Add Watch уточняются детали, касающиеся добавляемых контролируемых выражений.

В поле Expression (Выражение) указывается выражение, подлежащее контролю. Если перед открытием окна было выделено нужное выражение, менять в окне ничего не придется, но вы всегда имеете возможность подправить выражение.

В разделе Context (Контекст) можно определить процедуру, в которой редактор Visual Basic будет вычислять и отображать значение переменной. Изначально в полях Procedure (Процедура) и Module (Модуль) будут указаны процедура и модуль, из которых вы добавляете контролируемое выражение. Если вы хотите видеть значение при выполнении другой процедуры, выберите ее имя. Если это процедура из другого модуля, то сначала выберите нужный модуль, а уж затем процедуру.

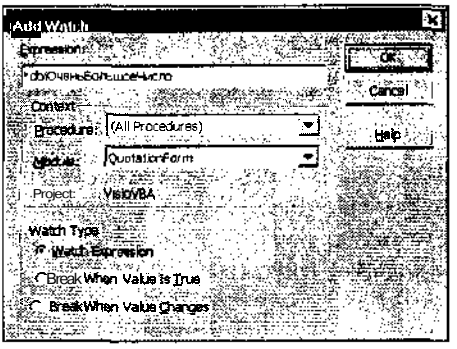


Рис. 9.13. Диалоговое окно Add Watch

Если вы хотите, чтобы значение переменной присутствовало в окне, независимо от того, какая процедура данного модуля выполняется, выберите (All Procedures) (Все процедуры), первый пункт в списке процедур. А если вы хотите видеть значение всегда — в течение всего времени выполнения программы — в списке модулей выберите (All Modules) (Все модули).

Для обычных контролируемых выражений в разделе Watch Type (Тип контроля) оставьте выбранным переключатель Watch Expression (Контролируемое выражение). Оставшиеся две возможности для выбора будут обсуждаться немного позже, в разделе "Использование контролируемых выражений для назначения точек останова",

Редактирование контролируемых выражений

Проще всего модифицировать существующее контролируемое выражение, внося изменения в столбец Expression (Выражение) окна Watches. Сначала щелкните в строке нужного вам выражения, чтобы выделить ее, а затем щелкните на самом выражении, чтобы выделить его. После этого можете менять свое выражение на любое другое.

Как и в окне Locals, в окне Watches тоже можно менять значение выражения прямо во время выполнения программы. А вот установки в столбце Context для контролируемого выражения в окне Watches менять нельзя. Для этого нужно открыть диалоговое окно Edit Watch (Редактирование контролируемого выражения), предлагаемое меню Debug или контекстным (открывается щелчком правой кнопки мыши) меню в окне Watches. В этом диалоговом окне вы сможете изменить контекст контролируемого выражения, его тип и даже само выражение точно так же, как в окне Add Watch при первоначальном добавлении выражения к списку контролируемых,

Использование контролируемых выражений для назначения точек останова

По мере увеличения размеров программы становится все труднее следить за изменениями значений переменных различными операторами и процедурами. Иногда вы видите, что в конечном итоге значение переменной оказывается неправильным, но не представляете, где именно в программе нужно искать источник проблем. В таких случаях спасательным кругом (по крайней мере, для спасения программного кода) оказываются точки останова, связанные с контролируемыми выражениями.

С помощью переключателей из раздела Watch Type (Тип контроля) в окнах **Add Watch** и **Edit Watch** вы получаете возможность назначить автоматические точки останова следующих двух типов.

- ✓ Точки останова, автоматически переводящие выполнение программы в режим паузы, как только программа выполнит оператор, изменяющий на ненулевое значение заданного выражения (напомню, что в VBA 0 означает False, а любое ненулевое значение — это True). Чтобы назначить автоматическую точку останова этого типа, выберите переключатель Break When Value Is True (Стоп при значении Истина).
- ✓ Точки останова, автоматически переводящие выполнение программы в режим паузы, как только программа выполнит оператор, изменяющий значение заданного выражения. Чтобы назначить автоматическую точку останова этого типа, выберите переключатель Break When Value Changes (Стоп при изменении значения).

Несмотря на свои выдающиеся таланты, "останавливающие" контролируемые выражения в окне Watches выполняют все, что требуется от обычных контролируемых выражений. Отличить контролируемые выражения разных типов можно только по маленьким пиктограммам слева от имен этих выражений. На рис. 9.14 эти пиктограммы различимы с трудом, но на экране они видны довольно хорошо.

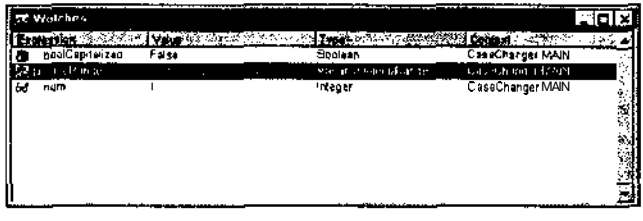


Рис. 9.14. Контролируемое выражение типа "стоп при изменении значения" на основе переменной caseRange только/сочтопереведет выполнение программы в режим паузы

После того как изменение значения контролируемого выражения активизирует связанную с этим выражением точку останова, VBA приостановит выполнение программы, переведя ее в режим паузы. В окне редактирования программного кода оператор, вызвавший изменение, будет предшествовать подсвеченному. В окне Watches подсвечено контролируемое выражение, вызвавшее переход в режим паузы, так что вы увидите, значение какого из контролируемых выражений изменилось (см. рис. 14.14).

Отлов "диких" ошибок с помощью On Error и Err

Если в ходе выполнения программы что-то идет не так, как нужно, обычно на работе программы это сказывается катастрофически. VBA прекращает выполнение программы и выводит диалоговое окно с немногословным сообщением об ошибке выполнения. Пример такого сообщения вы уже видели на рис. 9.2. На выбор вам предлагается щелкнуть либо на кнопке End (чтобы завершить выполнение программы), либо на кнопке Debug (чтобы перейти в режим паузы), либо на кнопке Help (чтобы отобразить на экране раздел справки, соответствующий произошедшей в вашей программе ошибке).

Ни одно из этих предложений не выглядит вполне удовлетворительным, особенно при выполнении программы в процессе эксплуатации, а не отладки. Менее раздражающей оказывается ситуация, когда ваша программа может сама распознать ошибку и исправить или, по крайней мере, мягко обойти ее прежде, чем VBA выскочит со своим грубым сообщением об ошибке. А если сделать это нельзя, то неплохо хотя бы одарить пользователя немного более дружелюбным сообщением по этому поводу.

Все это можно реализовать, но для этого вам придется добавить в программу свой собственный обработчик ошибок. О том, как это сделать, и пойдет речь в данном разделе.

Откуда берутся ошибки выполнения

Как я уже упоминал — но повторяю здесь, поскольку это было слишком давно, — появление ошибок выполнения обусловливается следующими двумя причинами.

- ✓ Наличие ошибок в логике программы, вследствие которых возникает программная ситуация, недопустимая с точки зрения VBA. Представьте, например, что в программном коде имеется оператор, в котором некоторое значение делится на значение переменной. Если перед этим переменной в программе будет присвоено нулевое значение — вы попались. Делить на нуль категорически запрещено, от этого у компьютера может случиться удар, и поэтому VBA остановит выполнение программы.
- ✓ Возникновение непредвиденного стечения обстоятельств, не позволяющего программе продолжить работу. Например, разрыв соединения с базой данных, сетевым сервером или страницей Internet может не позволить программе получить необходимые для ее работы значения.

Но независимо от причин, выход один — организация защиты программы и пользователя с помощью создания программного обработчика ошибок.

Как работает программный обработчик ошибок

Чтобы VBA не имел дела с ошибками выполнения, а взять заботу о них на себя, процедуры нужно снабдить *обработчиком ошибок* — блоком программного кода, выполняемым только тогда, когда происходит ошибка. Если при выполнении процедуры что-то пойдет не так, VBA вызовет этот обработчик ошибок, чтобы в нем выяснить, какого типа ошибка произошла, и в зависимости от сложившейся ситуации предпринять предусмотренные вами действия.

Как создать обработчик ошибок

Чтобы добавить в процедуру программный код обработчика ошибок, нужно сделать следующее:

- ✓ добавить в начало процедуры оператор On Error, сообщающий VBA, где найти программный код обработчика ошибок;
- ✓ в конец процедуры добавить оператор Exit Sub (или Exit Function), за которым должен следовать программный код обработчика ошибок;
- ✓ создать программный код самого обработчика ошибок, начало которого должно идентифицироваться некоторой меткой.

Все три шага при этом обязательны. Например, без оператора On Error VBA не узнает, что обработчик ошибок вами вообще предусмотрен.

Использование оператора On Error

Оператор On Error делает обработчик ошибок доступным, сообщая VBA о том, где найти этот обработчик ошибок в программном коде. В использовании этот оператор выглядит так: On Error GoTo *метка,* где *метка* — это метка, идентифицирующая в процедуре начало программного кода обработчика ошибок.

Вот пример использования оператора On Error:

```
Sub ErrorHandlerDemo()
On Error GoTo ErrorHandler
     MamaVariable = DoThisFunction (X,Y,Z)
     PapaVariable = DoThatProcedure
     EabyVariable = MamaVariable + PapaVariable
' здесь завершается выполнение процедуры, когда ошибок нет
     Exit Sub
ErrorHandler:
(здесь размещаются операторы обработчика ошибок)
End Sub
```

Вам полезно знать, что существуют еще две формы оператора On Error.

- ✓ On Error GoTo 0 отменяет программную обработку ошибок в процедуре, начиная с данного места, и выполняет очистку объекта Err. Используйте этот оператор, чтобы отменить действие ранее выполненного оператора On Error GoTo *метка*, если по каким-то причинам вы предпочтете отказаться от услуг своего обработчика ошибок в пользу "крикливо бесцеремонного" обработчика VBA.
- ✓ On Error Resume Next дает указание VBA игнорировать строку программного кода, ставшую причиной ошибки, и продолжить выполнение со следующей за ней строки. Ясно, что игнорирование ошибки не устраняет ее, но оператор On Error Resume Next оказывается полезным при работе с объектами из содержащего программу приложения или других компонентов, которые создали не вы, — эти достаточно тонкие вопросы обсуждаются в главе 14.

Добавление в процедуру оператора Exit

Если обработчик ошибок является частью той процедуры, в которой он размещен, VBA будет выполнять операторы обработчика ошибок при каждом выполнении процедуры. Ясно, что этого быть не должно. Нужно, чтобы обработчик ошибок выполнялся только тогда, когда в программе возникает ошибка.

Чтобы обработчик ошибок не выполнялся, когда это не нужно, необходимо добавить перед началом программного кода обработчика ошибок оператор Exit, как показано выше в примере из подраздела "Использование оператора On Error". Если это процедура типа Sub, то используйте оператор Exit Sub, а для процедуры типа Function используйте Exit Function.

Созданиеобработчикаошибок

Сам по себе обработчик ошибок состоит из следующих компонентов:

- ✓ метка, показывающая, где начинается программный код обработчика ошибок;
- ✓ программный код, призванный определить тип возникшей ошибки;
- ✓ программный код, обрабатывающий ошибку;
- ✓ оператор, дающий указание продолжить выполнение основной части процедуры (необязательный компонент).

Добавлениеметки

Для идентификации начала программного кода обработчика ошибок поместите в первой строке обработчика ошибок стандартную метку VBA (напомню, что метка представляет собой любое допустимое слово, за которым следует двоеточие). Как уже объяснялось выше, в случае возникновения ошибки этой метке должен передать управление оператор On Error. В следующем примере такой меткой является ThisIsTheErrorHandler:

```
...
ThisIsTheErrorHandler:
...
      (здесь размещаются операторы обработчика ошибок)
...
End Sub
```

Программный код обработчика ошибок почти всегда размещают в самом конце процедуры, поэтому и в данном примере обработчик ошибок завершается оператором End Sub.

Получениеинформацииобошибкеиееобработка

Перед обработчиком ошибок ставятся две главные задачи — выяснение типа возникшей ошибки и выполнение некоторых действий по ее исправлению. По сути, эти задачи различны, но на практике они выполняются почти одновременно.

Как правило, при выяснении того, что произошло, используется объект Err VBA. Объект Err всегда доступен в любой VBA-программе. Его можно использовать сразу, без предварительного создания его экземпляра. В объекте Err VBA автоматически сохраняет информацию о последней из случившихся в программе ошибок. Вашей программе остается только выяснить значения свойств объекта Err, в частности значения свойств Number (Номер) и Description (Описание).

Свойство Number сообщает номер ошибки. Вы можете присвоить значение этого свойства переменной или использовать его непосредственно в условном выражении, как в следующем примере:

```
Sub ЕщеОдноСообщение()
On Error GoTo ПытаюсьДуматьНоНеВыходит
...
 (склонные к ошибкам операторы)
...
Exit Sub
ПытаюсьДуматьНоНеВыходит:
Select Case Err.Number
     Case 7  ' выход за пределы памяти
           (обработка ошибки номер 7)
     Case 11 ' деление на нуль
           (обработка ошибки номер 11)
     ...  (и т.д.)
```

```
    Case Else
        (обработка всех остальных ошибок)
End Select
End Sub
```

В этом примере оператор Select Case сравнивает значение свойства Number с рядом фиксированных значений, для каждого из которых нужно создать обрабатывающий соответствующую ошибку фрагмент программного кода. Обработка ошибки может включать следующее.

- ✓ Информирование пользователя о том, что произошло, и получение от пользователя инструкций по этому поводу с помощью окна ввода или другой специальной формы.
- ✓ Попытка получить данные снова из того же или альтернативного источника.
- ✓ Изменение неподходящих данных на подходящие с одновременной записью в файл или выводом на экран сообщения о том, что для продолжения работы пришлось эти данные изменить.

Для эффективной работы со свойством Number нужно знать, каким именно ошибкам соответствуют возвращаемые этим свойством номера. Из-за недостатка места я не могу представить здесь соответствующий список. В некоторых VBA-приложениях — к ним относятся и приложения Office 97 — информация об ошибках, встречающихся чаще всего, имеется в разделе Trappable Errors (Распознаваемые ошибки) справки по VBA. Этот раздел можно найти, напечатав trappable errors в поле ввода на вкладке Предметный указатель в окне справки. В справочной системе Office 2000 подобного раздела мне найти не удалось.

Даже если ваш программный код не может обработать какую-то ошибку или если вы просто не хотите утруждать себя, вы все равно можете не допустить появления стандартного сообщения VBA об ошибке, использовав свойство Description (Описание) объекта Err. Следующий фрагмент программного кода отображает исключительно вежливое сообщение об ошибке, частью которого является строка, предоставленная свойством Description:

```
strMoeCooбщение = "С сожалением сообщаю Вам, что " _
     & "в эту в целом замечательную программу все же " _
     & "вкралась ошибка. Как сообщает наш надежный " _
     & "источник (VBA), причиной беспокойства является "
MsgBox strMoeCooбщение & Err.Description
```

Некоторые VBA-приложения предлагают свои объекты и функции, с помощью которых можно получить дополнительную информацию о специфических проблемах данного приложения. Например, Microsoft Access предлагает для этого специальный объект Error.

Продолжение выполнения программы

После того как обработчик ошибок завершит свою работу, вам придется решить, вернуться ли в процедуру, в которой произошла ошибка, чтобы продолжить выполнение этой процедуры, или передать управление той процедуре, из которой была вызвана процедура с ошибкой.

Если вы предпочтете продолжить выполнение текущей процедуры, поместите в обработчик ошибок оператор Resume. Оператор Resume имеет несколько вариантов.

- ✓ Если просто напечатать Resume в отдельной строке, то этот оператор вернет управление в процедуре тому же оператору, который вызвал ошибку. Это разумно, если обработчик ошибок решает проблему и вы уверены, что ошибка не возникнет вновь.

- ✓ Чтобы пропустить оператор, ставший причиной появления ошибки, используйте в обработчике ошибок оператор Resume Next. Тогда выполнение продолжится с оператора, следующего непосредственно за оператором, вызвавшим ошибку.
- ✓ Чтобы продолжить выполнение с определенного места в процедуре, используйте в обработчике ошибок оператор Resume *метка*. Здесь *метка* означает ссылку на метку, размещенную где-то в процедуре, но, очевидно, *не* на метку, идентифицирующую начало программного кода обработчика ошибок. Ясно, что в данном случае вы должны задать соответствующую метку в процедуре.

Глава 10

Создание интерактивных VBA-форм

В этой главе...

- Запуск и печать форм в процессе проектирования
- Создание новой формы и размещение в ней элементов управления
- Использование окна свойств — простейший способ установки свойств в VBA
- Обзор важнейших свойств форм и элементов управления
- Работа с подписями, текстовыми полями, кнопками и рамками
- Отображение форм в VBA-программе и удаление их с экрана
- Создание процедур обработки событий, обеспечивающих интерактивное взаимодействие с формами

С помощью VBA можно создавать очень привлекательные на вид окна, будь то панели управления, расширяющие возможности приложений, или простые вспомогательные диалоговые окна. По сравнению с созданием программного кода (или даже просто с печатанием приличного куска текста) создание пользовательской формы вообще может казаться игрою. Все же кое-какую работу при этом нужно выполнить и в некоторых случаях неплохо кое-что знать, чтобы не попасть в тупик. В этой главе рассказывается об основных шагах процесса создания формы: планирование, создание пустой формы в редакторе Visual Basic, добавление в форму элементов управления и работа со свойствами формы и элементов управления. Добавлять элементы управления в формы несложно, а вот чтобы заставить их делать то, что вам нужно, потребуется немного больше умственной работы и программирования. В этой главе мы также обсудим и тонкости процесса программирования форм.

Основы проектирования форм

Перед тем как на практике заняться проектированием форм, нелишне обсудить некоторые детали самого процесса. Знание этих деталей может ускорить процесс и уберечь от неверных решений.

Запуск форм

В процессе разработки формы ее можно запустить (т.е. активизировать выполнение и отобразить на экране) в любое время. Для этого выполните следующее.

1. **Выберите окно формы.**
   Щелкните в окне формы или в окне программного кода, ассоциированного с формой.
2. **Отобразите форму.**
   Нажмите <F5> или щелкните на кнопке Run (Выполнить).

После этих действий ваша форма появится на фоне VBA-приложения (а окно Visual Basic пока скроется из виду). Вы сможете проверить, что происходит после щелчков на форме и ее элементах управления. При этом не забывайте, что если ваша форма входит как часть в другую программу, то эта форма при ее автономном запуске может работать не так, как она работала бы в рамках выполнения содержащей ее программы.

Во время запуска окно формы должно быть выбранным. В редакторе Visual Basic маркеры выделения по периметру формы остаются видимыми и при переключении в другое окно редактора Visual Basic. Форма может казаться выделенной, но при этом не запускаться нажатием клавиши <F5>. Так что если вдруг на экране неожиданно возникнет диалоговое окно Macros (Макросы), просто щелкните сначала в нем на кнопке Cancel (Отмена), а затем в любом месте нужной вам формы, прежде чем повторить попытку ее запуска.

Чтобы прекратить выполнение формы, которую вы еще не снабдили кнопкой Отмена, щелкните на ее закрывающей кнопке, находящейся с самого края, справа в строке заголовка формы. Или нажмите <Alt+Tab> и переключитесь в окно редактора Visual Basic, где можно щелкнуть на кнопке Reset (Остановить выполнение) панели инструментов.

Чтобы запустить форму в автономном режиме, вы должны находиться в окне редактора Visual Basic и использовать только что описанные приемы. Но форма создается для ее запуска из VBA-приложения, для чего вам нужно активизировать форму из программного кода, как рассказано дальше в настоящей главе, а сам программный код выполнить с помощью приемов, обсуждавшихся в главе 4.

Формы и элементы управления — программируемые объекты

Напомню еще раз, что формы и элементы управления — это полноценные объекты VBA. со своими свойствами, методами и событиями. Формы и элементы управления отличаются от других объектов лишь одним; вы можете менять их свойства в диалоговых окнах без программирования. Правда, для каждого элемента управления (за исключением некоторых надписей и фреймов) вам все же потребуется написать некоторый программный код. Да и формы тоже часто требуют программирования. Не забывайте об этом при обдумывании плана работ по своему проекту. Этап программирования в процессе создания форм обсуждается дальше в настоящей главе.

Один момент, касающийся форм в Microsoft Access: хотя для создания VBA-программ в Access и используется стандартный язык VBA, а формы Access со своими элементами управления выглядят и работают точно так же, как и в других VBA-приложениях, формы Access все же не являются стандартными VBA-формами. Свойства элементов управления в Access отличаются от свойств стандартных элементов управления VBA, поэтому формы, созданные в Access, и формы, созданные в других VBA-приложениях, не взаимозаменяемы.

Планирование форм для программы

Создавать формы в VBA легко и просто, но их проектирование в реальной программе требует тщательного планирования. Не забывайте, что формы являются частью большой программы, которая имеет вполне определенное практическое назначение. Поэтому, прежде чем начать забавляться с формами, уделите хотя бы немного времени следующим шагам.

1. **Определите цель для своей программы.**
   Это потребуется, даже если в программе нет форм вообще, но если формы нужны, то при их создании всегда следует помнить о главной цели всей программы.
2. **Для каждой из форм в программе определите ту специальную задачу, которую должна решать данная форма в русле главной задачи программы.**
   Какую информацию должна отображать форма, какая информация потребуется от пользователя и какие данные потребуются пользователю для того, чтобы решить, в какое русло направить выполнение программы дальше? Ответы на эти вопросы вы должны трансформировать в конкретные элементы формы.
3. **Для каждой специальной задачи подберите тот элемент управления, который лучше всего подходит для ее решения.**
   Вы собираетесь попросить пользователя выбрать одну из нескольких взаимоисключающих возможностей? Тогда используйте кнопки переключателей. Но если возможности независимы, то лучше всего использовать флажки.
4. **Решите, где именно разместить каждый из элементов управления в форме.**
   Критериями принятия решения должны быть важность выполняемой элементом управления задачи, вероятная частота использования, связь с другими выполняемыми задачами (чтобы знать, с какими другими элементами управления сгруппировать данный).
5. **Выяснив все эти детали, сделайте (хотя бы мысленно) набросок общего вида своих форм.**
   Не размещайте в формах слишком много элементов управления. Разумное использование форм с множеством страниц уменьшает обшее число форм, но будьте готовы также добавить в проект формы, если это поможет избежать беспорядка и упростить использование форм.
6. **После того как вы учтете все вышеописанные практические детали, можете подумать об эстетических элементах.**
   Будет ли особый шрифт на кнопках или стрелки в виде ракет на кнопках прокрутки помогать пользователю работать с формой? В разумных пределах правильно подобранные цвета, шрифты и изображения, несомненно, привлекают интерес, но неумеренность здесь вызовет только раздражение.

Я не говорю, что вы должны потратить полжизни на абстрактное планирование. Просто даже общее представление о том, что нужно получить, поможет в разработке пробной версии и тестировании ее в надежде выяснить, что в конце концов будет работать, а что — нет. И я хотел бы обратить ваше внимание на то, что при полном игнорировании планирования впоследствии наверняка потребуется немало усилий для переделки созданных форм.

Печать форм в процессе проектирования

При разработке форм для VBA-проекта иногда полезно напечатать копии форм, которые всегда будут под рукой даже там, где нет компьютера. Распечатки набросков форм годятся и для их критического редактирования, и для предъявления их потенциальным пользователям или коллегам-программистам для выяснения соответствующих мнений.

Чтобы распечатать формы проекта в редакторе Visual Basic, выполните следующее.

1. **Если нужно распечатать только одну форму, выделите ее в окне проводника проектов, если же нужно распечатать все формы некоторого проекта, выделите форму, модуль или любой другой компонент этого проекта.**
2. **Выберите File⇨Print или нажмите <Ctrl+P>.**

3. **В появившемся диалоговом окне отметьте флажок Form Image (Изображение формы)-**
4. **Снимите флажок Code (Программный код), если вы не желаете распечатывать вместе с изображением формы ее программный код.**
5. **Выберите Current Module (Данный модуль), если нужно распечатать только одну форму, или Current Project (Данный проект), если нужно напечатать все формы проекта.**
6. **Щелкните на кнопке OK.**

Дизайн новой формы

Только что созданная новая форма (рис. 10.1) представляет собой чистое поле для ваших упражнений в разработке пользовательского интерфейса. Вы можете менять размеры формы и ее положение на экране, подбирать форме цвета и размещать в ней элементы управления, которые будут выполнять важные задачи.

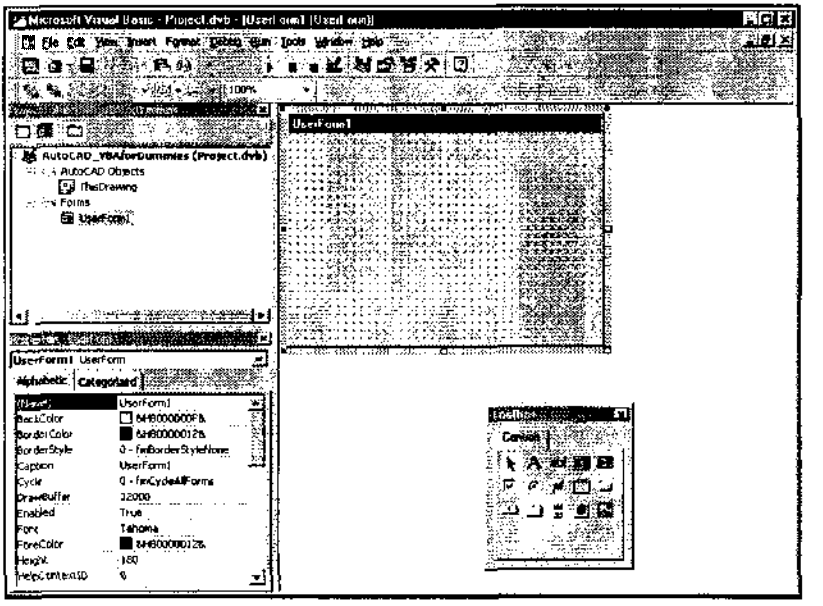


Рис. 10.1. Только что созданная форма (обратите внимание на панель элементов управления возле нее и пиктограммы элементов управления в этой панели)

Перед созданием новой формы убедитесь, что активен именно тот проект, в котором должна разместиться создаваемая форма. Сначала откройте документ проекта в VBA-приложении, затем переключитесь в редактор Visual Basic. Если приложение позволяет открыть несколько документов сразу или если для документа используется шаблон, то для выбора нужного проекта необходимо использовать проводник проектов. (О работе с проводником проектов говорилось в главе 5.)

Создание формы

Для создания новой формы VBA выберите Insert⇨UserForm из меню редактора Visual Basic или из контекстного меню (вызываемого щелчком правой кнопки мыши) в окне проводника проектов. Новая форма (точнее, заготовка новой формы) появится в специально созданном для нее ок-

не, а возле формы появится панель элементов управления (см. рис. 10.1) — специальная панель инструментов, содержащая элементы управления, которые можно разместить в этой форме.

Добавление элементов управления из панели инструментов Toolbox

Теперь заготовка формы на экране, и можно добавлять в нее элементы управления — всякие разные штучки в форме, с помощью которых взаимодействуют люди. Элементы управления берутся из панели элементов управления (панели **Toolbox).**

Панель элементов управления автоматически появляется при создании новой формы, если только перед этим вы не закрыли эту панель. Панель элементов управления грациозно исчезает с экрана, когда вы щелкаете в любом из окон, не содержащих формы, и возникает снова после щелчка в окне формы. Если же панель элементов управления не видна, когда она вам понадобилась, выберите View⇨Toolbox, чтобы вызвать ее.

Чтобы добавить в форму элемент управления, выполните следующее.

1. **Щелкните на пиктограмме того элемента управления в панели элементов управления, который вы хотите поместить в форму.**
2. **Переместите указатель мыши на форму.**
   Указатель мыши примет вид крестика с рисунком выбранного элемента управления.
3. **Нажмите левую кнопку мыши и перетащите указатель, очертив область на форме, которую должен занять новый элемент управления (рис. 10.2).**
4. **Отпустите кнопку мыши.**
   В форме должен появиться новый элемент управления.

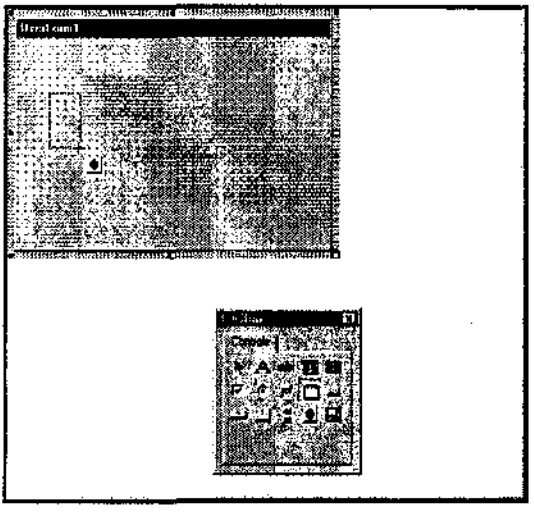


Рис. 10.2. Добавление элемента управления в форму

Обычно сразу же после добавления элемента управления в форму указатель мыши приобретает свой обычный вид, позволяющий выбирать объект. Если вам нужно добавить несколько элементов управления одного и того же типа, вместо одного щелчка на пиктограмме соответствующего элемента управления в панели элементов управления сделайте двойной щелчок. Чтобы закончить добавление элементов управления, щелкните на пиктограмме стрелки в панели элементов управления.

Работа со свойствами форм и элементов управления

Формы и элементы управления VBA являются объектами, а поэтому имеют свойства, задающие вид и поведение форм и элементов управления на экране. Эти свойства, как и свойства любых других объектов в программе, можно проверить и изменить.

Но для объектов, о которых речь идет здесь, вам *не нужно* писать программный код самим. Редактор Visual Basic предлагает воспользоваться окном свойств, которое позволяет с легкостью управлять множеством важных характеристик форм и элементов управления, не программируя их. Правда, по мере усложнения ваших программ вам все чаще и чаще придется устанавливать значения свойств программно, но и тогда окно свойств останется лучшим выбором для установки начальных значений свойств форм и элементов управления в них,

Общий вид окна свойств показан на рис. 10.3. Если окно свойств на экране отсутствует, оно появится после выбора View⇨Properties Window из меню или нажатия <F4>.

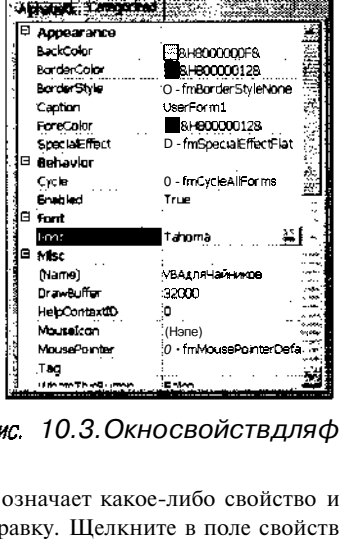
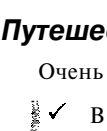


Рис. 10.3. Окно свойств для формы VBA



Когда не ясно, что означает какое-либо свойство или как установить его значение, просто вызовите справку. Щелкните в поле свойства и нажмите <F1>, чтобы на экране появилось окно справки с текстом соответствующего раздела справки VBA,

Путешествие в окне свойств

Очень удобны следующие особенности окна свойств.

- ✓ В окне свойств автоматически отображаются свойства того объекта, который выбран в окне формы (это может быть как элемент управления, так и сама форма).
- ✓ Выбрать элемент в форме для работы с ним можно из раскрывающегося списка вверху окна свойств (рис. 10.4).

Обратите внимание на то, что в поле списка в верхней части окна показаны имя и тип того объекта, со свойствами которого вы имеете дело в данный момент.

В окне свойств есть две вкладки, на которых отображается один и тот же набор свойств, но на вкладке Alphabetic (По алфавиту) свойства упорядочены по размеру годового дохода (шутка), а на вкладке Categorized (По категориям) они классифицированы по родственным признакам.

Какую бы из вкладок вы ни выбрали, работа в окне свойств принципиально не меняется. Слева будут указаны имена свойств, а справа от каждого имени — *поле,* в котором вы сможете изменить текущую установку соответствующего свойства.

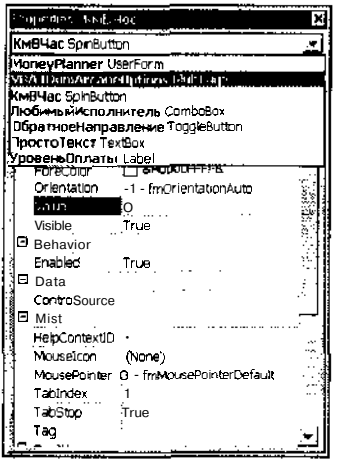


Рис. 10.4. С помощью раскрывающегося списка вверху окна свойств можно выбрать любой элемент управления в форме или саму форму, свойства которых нужно увидеть или поменять

Изменение установок свойств

В зависимости от свойства, установки свойств можно менять одним из следующих трех способов:

- ✓ просто печатая новое значение;
- ✓ выбирая новое значение из раскрывающегося списка;
- ✓ выбирая новое значение в диалоговом окне.

Некоторые свойства позволяют как напечатать новое значение, так и выбрать подходящее из списка.

До тех пор пока вы не щелкнете в поле свойства, нельзя сказать, предлагается ли для этого свойства раскрывающийся список или диалоговое окно. Если для свойства предлагается список или диалоговое окно, после щелчка в поле свойства появится кнопка, на которой можно щелкнуть. (В случае раскрывающегося списка на кнопке будет направленная вниз стрелка, а в случае диалогового окна — многоточие; см. рис. 10.3.)

Не забывайте выбрать нужный элемент

Пока элементы управления не добавлялись в форму, в раскрывающемся списке окна свойств будет представлена только сама форма. Но после того, как хотя бы один элемент управления добавлен, вам придется следить за тем, какой из объектов выбран (сама

форма или один из элементов управления). В VBA это совсем нетрудно — по углам и сторонам выбранного объекта появляются маленькие квадратики *маркеров*.

Конечно, чтобы выделить объект, можно просто щелкнуть на нем. Но иногда это оказывается не слишком простым делом.

С одной стороны, даже если окно формы у вас прямо перед глазами, вполне обычной оказывается ситуация, когда вы думаете, что выбрали нужный объект, переходите в окно свойств и начинаете изменять свойства, а потом выясняется, что выбран не тот объект. Лично у меня нет хороших рекомендаций по поводу того, как этого избежать. С другой стороны, если форма перегружена элементами управления или (особенно) если вы разместили одни элементы управления на других, то щелкнуть на нужном элементе управления совсем непросто.

По этим причинам я рекомендую всегда выбирать объекты в форме из раскрывающегося списка вверху окна свойств. Маркеры вокруг выбранного объекта будут появляться при этом точно так же, как и после щелчка на этом объекте. Если же вы предпочитаете выбирать щелчком, возьмите за правило после щелчка хотя бы взглянуть на имя в поле раскрывающегося списка в окне свойств, чтобы убедиться в выборе нужного объекта.

Ключевые свойства форм

Многие свойства форм и элементов управления оказываются одинаковыми и используются одинаково. Например, одинаково задаются размеры как любого элемента управления, так и самой формы. Одинаково задается как цвет фона для формы, так и цвет фона для текста на кнопках. В этом разделе мы обсудим наиболее важные из относящихся к форматированию свойств, общие для всех элементов управления и форм.

При создании формы возьмите за правило устанавливать следующие свойства формы еще до того, как в форму будут добавлены элементы управления.

Необходимое действие	Соответствующее свойство	Замечание
Задание имени формы	(Name)	Имя используется для ссылок на форму в программном коде. Выберите для формы достаточно короткое имя, но ясно указывающие на ее функциональное назначение
Изменение заголовка формы	Caption	Свойство Caption (Заголовок) содержит текст, который будет появляться в строке заголовка формы. Для заголовка выберите текст, который будет напоминать пользователю о назначении формы
Необходимое действие	Соответствующее свойство	Замечание
Выбор шрифта для использования по умолчанию	Font	

Необходимое действие	Соответствующее свойство	Замечание
Выбор используемого по умолчанию цвета текста для элементов управления в форме	ForeColor	
Выбор цвета фона формы	BackColor	

В следующих разделах эти свойства обсуждаются подробнее.

Имя и заголовок — не одно и то же

Как вы уже знаете, каждый объект в VBA-программе имеет имя, и формы и элементы управления не исключение. Имя формы не появляется на самой форме. Имя используется для ссылок на объект в программном коде, например, когда требуется активизировать в программе один из методов объекта или изменить одно из свойств объекта.

Формы и элементы управления действительно имеют свойство Name (Имя). Если задуматься, здесь есть некоторое противоречие. С одной стороны, для ссылки на объект нужно его имя. Но если для доступа к формам из коллекции Forms или к элементам управления в форме использовать цикл For Each, то программа получит ссылку на объект, и не зная его имени.

В любом случае, главное здесь то, что задаваемое вами в свойстве Name имя будет именем объекта, которое вы используете в своем программном коде, когда захотите что-либо сделать с этим объектом. Имя формы или элемента управления должно быть допустимым в VBA именем (никаких пробелов и знаков пунктуации; см. главу 6). Кроме того, имя должно описывать функциональное назначение объекта в программе.

А *заголовок* формы или элемента управления — это нечто другое. Заголовок формы представляет собой текст, появляющийся в ее строке заголовка. Заголовок же элемента управления — если для этого элемента управления заголовок вообще предусмотрен, — это текст надписи, появляющийся на этом элементе управления. Свойство Caption (Заголовок) можно изменить, впечатав новый текст в соответствующее поле окна свойств. В случае элементов управления можно сначала щелкнуть на элементе управления в форме, чтобы выделить его, а затем щелкнуть на нем еще раз и редактировать заголовок прямо на элементе управления. (Не используйте при этом двойной щелчок — тогда откроется окно редактирования программного кода.)

Модальные формы против немодальных

Классическим примером *модальной* формы является обычное диалоговое окно: пока диалоговое окно открыто, вы не можете работать с другими частями программы; чтобы работать с другим окном или с самим документом, вы должны сначала закрыть диалоговое окно. Точно так же, когда активна модальная форма, в VBA могут выполняться только те процедуры, которые принадлежат этой форме. Форма может реагировать на события, но остальная часть программы остается "замороженной", равно как и само приложение, в котором выполняется программа, остается недоступным пользователю, пока эта форма не будет закрыта.

В VBA 6.3 (точно так же, как и в VBA 6) вы можете назначить любому диалоговому окну модальное или *немодальное* поведение. По умолчанию для соответствующего свойства ShowModal формы установлено значение True, и это значит, что форма будет модальной.

Если нужно обеспечить пользователю возможность переключаться от этой формы к другим открытым формам и обратно, выберите для этого свойства значение False. В этом случае ваша программа — та, которая отобразит немодальную форму, — продолжит активно выполняться и тогда, когда форма будет на экране.

Пользователи программы получают возможность перемещаться от формы к форме только тогда, когда *все* открытые в данный момент формы определены как *немодальные.* Как только программа отобразит модальную форму, выполнение остальной части программы приостановится, пока пользователь не закроет эту форму. Если перед отображением модальной формы на экране присутствовали другие отображенные программой формы, эти формы останутся на экране видимыми, но их нельзя активизировать до тех пор, пока модальная форма не будет закрыта.

Хотя эта книга посвящена преимущественно VBA 6, если вы работаете с более ранними версиями VBA, вам следует знать, что в VBA 5 все формы модальны — свойства ShowModal у них нет.

Изменение размеров и позиции форм и элементов управления

Перед тем как приступить к более или менее серьезному проектированию форм, я считаю нужным сообщить вам, что, как и любая серьезная графическая программа, редактор Visual Basic имеет координатную сетку. Эта *сетка* составлена из воображаемых магнитных линий — вертикальных и горизонтальных. При перемещении или изменении размеров элементов в форме их стороны автоматически размещаются вдоль ближайшей линии сетки. Здесь я должен подчеркнуть, что сетка очень поможет в работе, если правильно выбрать ее параметры для разрабатываемого проекта.

Но вернемся к нашим баранам. Менять размеры форм и элементов управления совсем просто. Можно задать значения свойств Height (Высота) и Width (Ширина) в окне свойств, но еще проще захватить и перетаскивать один из маркеров изменения размеров объекта — один из маленьких квадратиков, появляющихся вокруг изображения объекта, когда последний выделен. Я понимаю, вы знаете, как изменять размеры объектов — любая программа в Windows (от Paint до чего угодно), позволяющая менять размеры графических объектов, использует именно этот метод. Но, на всякий случай, привожу формальное описание процедуры.

- ✓ Чтобы изменить размеры формы в одном измерении, перетащите нижний или правый *боковой* маркер.
- ✓ Чтобы изменить размеры формы в обоих измерениях, сразу перетащите *угловой* маркер.

Изменение *размеров с помощью мыши*

Обратите внимание на то. что для изменения размеров годятся только белые маркеры, а большинство маркеров формы — черные. Черные маркеры не делают ничего, только выделяют саму форму. Форма в своем окне всегда привязана к верхнему левому углу окна, но не беспокойтесь — позиция формы на экране от этого совершенно не зависит.

Изменение размеров непосредственным заданием числовых значений

Можно непосредственно задавать размеры формы или помещенных в нее элементов управления, меняя числовые значения свойств Height и Width в окне свойств, Значения в данном случае задаются в пунктах *(пункт* равен 1/72 дюйма).

Управление размещением формы на экране

Вы имеете полный контроль над тем, где именно на экране должна появляться форма при отображении ее программой (или при пробном запуске самой формы из окна редактора Visual Basic). Для этого предлагается использовать свойство `StartUpPosition`.

По умолчанию для этого свойства устанавливается значение `1-CenterOwner`. Это значит, что форма должна появиться в центре окна VBA-приложения — независимо от его размеров и положения на экране (правда, если форма при этом частично оказывается за пределами экрана, то она придвинется своим краем вплотную к краю экрана, но никак не дальше). Если нужно, чтобы форма всегда появлялась посередине экрана, независимо от того, где размещено окно VBA-приложения, выберите для свойства `StartUpPosition` значение `2-CenterScreen`. Чтобы установить свои собственные значения, выберите `0-Manual`, а после этого задайте значения свойств `Left` (Левый край) и `Top` (Верх).

Позиционирование элементов управления

Чтобы изменить положение элемента управления в форме, просто перетащите этот элемент управления туда, куда нужно. Если же вам нужна особая точность, напечатайте подходящие числовые значения для свойств `Left` (Левый край) и `Top` (Верх).

Шик специальных эффектов

Если вы хотите придать своим формам трехмерный вид, воспользуйтесь свойством `SpecialEffect`, имеющимся у форм и некоторых элементов управления. При выборе для этого свойства значения из раскрывающегося списка, отличного от 0 (соответствующего плоскому виду), объекту придается небольшая, но достаточно заметная глубина.

Работа с элементами управления

Перед тем как перейти к использованию профессиональных команд форматирования для элементов управления, давайте разберемся с основами. Мы уже обсудили выше, в разделе "Ключевые свойства форм", как менять размеры элементов управления и перемещать их с помощью мыши или посредством изменения значений свойств. В этом разделе мы рассмотрим другие простые приемы редактирования форм и элементов управления в них.

Вырезание, копирование, вставка

Как и все другие программы для Windows, редактор Visual Basic позволяет вырезать, копировать и вставлять элементы управления и по отдельности, и группами. При этом используются стандартные команды меню и стандартные комбинации клавиш. Кроме того, в вашем распоряжении кнопки Cut (Вырезать), Copy (Копировать) и Paste (Вставить) панели инструментов Standard (Стандартная) редактора Visual Basic. (Эти кнопки выглядят точно так же, как аналогичные кнопки в Microsoft Office.)

Из этих команд дополнительных пояснений требует только команда `Paste`. Когда вы вставляете элемент управления из буфера обмена, VBA помещает элемент управления в центр формы, даже если эта часть формы не видна на экране. Но если перед операцией вклеивания выбрать фрейм или форму с множеством страниц, элемент управления будет размещен в центре соответствующего объекта.

Удаление элементов управления

Один или несколько элементов управления можно удалить, не помещая их в буфер обмена, если нажать клавишу <Delete> или выбрать Edit⇨Delete. Обратите внимание на то, что нажатие клавиши <Backspace> в данном случае не работает.

Выбор сразу нескольких элементов управления

Можно выделить группу элементов управления, а затем перемещать их, менять размеры, вырезать или применять иные команды форматирования как к одному целому. Очень удобно таким образом одновременно устанавливать одинаковые значения общим свойствам элементов управления. Для выделения сразу нескольких элементов управления используйте следующие приемы.

- ✓ Щелкните на пиктограмме стрелки в панели **Toolbox и,** перетаскивая указатель мыши, охватите прямоугольником выделения те элементы управления, которые нужно включить в группу выделенных. Если в прямоугольник выделения попадет хотя бы часть элемента управления, этот элемент управления окажется в группе выделенных.
- ✓ Щелкните на первом элементе управления в группе, а затем, нажав и удерживая клавишу <Shift>, щелкните на элементе управления в противоположной части области выделения. В результате будут выделены все элементы управления, находящиеся между теми двумя, на которых вы щелкали.
- ✓ Щелкните на элементе управления, удерживая при этом нажатой клавишу <Ctrl>. Элемент управления будет добавлен к группе выделенных или будет исключен из нее, в зависимости от того, был во время щелчка элемент управления выделен или нет.

Выделив группу элементов управления, можно перемещать их, менять размеры, вырезать или применять иные команды форматирования как к одному целому. Выделение группы элементов управления очень ускоряет работу, когда нужно установить одинаковые значения общим свойствам нескольких элементов управления.

Отмена изменений

Обычно можно отменить результаты последней команды форматирования с помощью команды Undo (Отмена), вызываемой нажатием клавиш <Ctrl+Z>. Однако отмена не действует после изменения размеров формы, а также для изменений, выполненных в окне свойств.

Использование координатной сетки

Сетка — это разметка из вертикальных и горизонтальных линий, "нанесенных" на формы. Сетка выполняет следующие функции.

- ✓ **Визуализация направляющих, вдоль которых размешаются элементы управления с помощью мыши.** Направляющие линии проходят через точки, на которые вы, наверное, обратили внимание при работе с формами.
- ✓ **Автоматическое выравнивание элементов управления по линиям сетки при перемещении или изменении размеров элементов управления с помощью мыши.** Независимо от вашего желания, при перемещении (или изменении размеров) элементов управления с помощью мыши стороны элементов управления "прилипают" к линиям сетки. Хотя это и ограничивает вас в гибкости, зато обеспечивает заметную согласованность при разметке формы.

Эти две функции работают независимо. Можно, например, сделать сетку невидимой, но оставить в силе автоматическое выравнивание или наоборот.

Задание параметров сетки

Чтобы установить параметры сетки, выберите Tools⇨**Options** из меню. В появившемся диалоговом окне Options (Параметры) щелкните на вкладке General (Общие), чтобы добраться до элементов управления, с помощью которых задаются параметры, о которых идет речь (рис. 10.5).

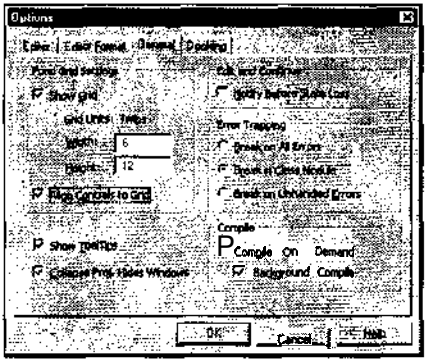


Рис. 10.5. Шаг сетки задается на вкладке General в диалоговом окне Options

В верхней левой части этого диалогового окна вы найдете несколько параметров, относящихся к сетке.

- ✓ **Show Grid** (Показывать сетку). Снимите этот флажок, если хотите, чтобы точек сетки на форме не стало. Состояние этого параметра не влияет на автоматическое выравнивание по линиям сетки.
- ✓ **Width** (Ширина) и **Height** (Высота). С помощью значений в этих полях текста задается шаг (т.е. расстояние между точками) сетки в вертикальном и горизонтальном направлениях соответственно.
- ✓ **Align Controls to Grid** (Выравнивать элементы управления по линиям сетки). Когда этот флажок отмечен, работает функция "прилипания" к линиям сетки. Сняв этот флажок, вы получите полную свободу в позиционировании и изменении размеров элементов управления. И повторяю, сетка *может* оставаться видимой, когда функция автоматического выравнивания отключена.

Форматирование элементов управления

В большинстве своем люди питают склонность к таким неосязаемым свойствам, как симметрия, согласованность и аккуратность. И вы, я думаю, хотели бы помочь пользователю сосредоточиться на тех задачах, которые предстоит выполнить, а не собираетесь вызвать у него раздражение беспорядочным нагромождением элементов управления.

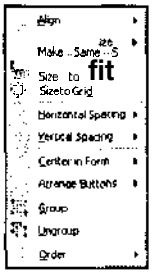


Рис. 10.6. Меню Format редактора Visual Basic

К счастью, в VBA есть все средства для создания хорошо организованных форм. Причем, хотя для этого все еще нужна некоторая ручная работа, средства автоматического форматирования VBA позволяют автоматизировать значительную часть процесса.

Использование меню Format

Команды, которые относятся к размещению и позиционированию элементов управления в форме, находятся в меню Format (Формат) редактора Visual Basic (рис. 10.6). Хорошее предварительное знакомство с пунктами этого меню и его многочисленных подменю сослужит вам хорошую службу во время проектирования формы.

Использование панели инструментов UserForm

При работе с формами в редакторе Visual Basic очень удобной оказывается панель инструментов UserForm. Если эта панель инструментов на экране не видна, отобразите ее, щелкнув правой кнопкой мыши в любой из видимых панелей инструментов и выбрав UserForm в появившемся меню. На рис. 10.7 панель инструментов UserForm показана как свободно перемещаемая панель.

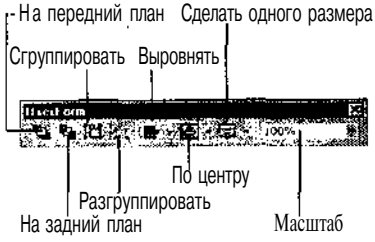


Рис. 10,7. Панель инструментов UserForm предлагает кнопки для часто используемых команд форматирований элементов управления

Кнопки панели инструментов UserForm соответствуют пунктам меню Format. Некоторые из этих кнопок комбинированные — справа от них видны небольшие направленные вниз стрелки, раскрывающиеся в списки опций. Если щелкнуть на главной части такой кнопки, VBA немедленно активизирует опцию, выбранную последней. Чтобы выбрать другую опцию, щелкните на маленькой стрелке и выберите опцию из появившегося списка.

Группировка элементов управления

Выбрать с помощью мыши сразу несколько элементов управления просто, но этот метод оказывается не идеальным в том случае, когда приходится постоянно работать с одним и тем же набором элементов управления как с отдельной единицей. Объединив все эти элементы управления в *группу,* вы избавляетесь от необходимости выбирать все эти элементы управления каждый раз, когда приходится с ними что-то делать, и тем самым исключаете возможные при таком выборе ошибки. На рис. 10.8 показаны примеры сгруппированных наборов элементов управления.

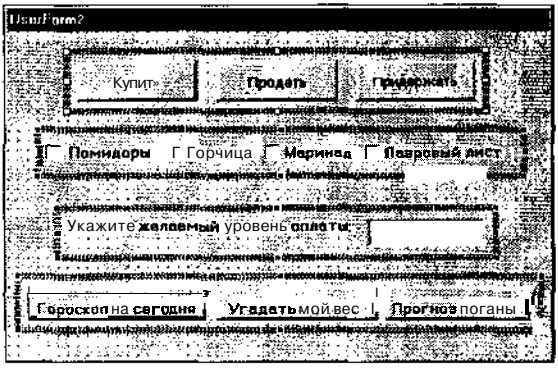


Рис. 10.8. Несколько групп элементов управления. Обратите внимание, что прямоугольники выделения охватывают все элементы управления в группе

Создать группу несложно.

1. **Выделите все** элементы **управления, которые должны войти в группу.**
2. **Щелкните** на **кнопке Group (Сгруппировать) панели инструментов UserForm или выберите** Format⇨Group из **меню.**

Группировка позволяет применять команды форматирования сразу ко всем элементам группы одновременно. Например, если необходимо выровнять расстояния между тремя рядами кнопок, сгруппируйте кнопки по рядам, выделите все три образовавшиеся группы и выберите Horizontal Spacing⇨Make Equal из меню (подробнее об этом — ниже в разделе "Выравнивание горизонтальных и вертикальных расстояний").

Размещение одних элементов управления поверх других

Хотя лучше всего не перекрывать одни элементы управления другими, это бывает необходимым, когда содержимое формы должно меняться в процессе выполнения программы. При этом, если правильно установить значения свойства Visible элементов управления, программа в каждый конкретный момент времени может оставлять невидимыми все перекрывающиеся элементы управления, кроме одного.

Но в окне редактора Visual Basic все элементы управления всегда видимы — разве что какой-нибудь из них полностью закрыт другими элементами управления в форме. Если такое случилось, используйте команды Order (Упорядочить) из меню Format, чтобы переупорядочить элементы управления.

Вот рекомендации по поводу использования команд Order.

- ✓ Если можно добраться хотя бы до небольшого кусочка спрятанного элемента управления, щелкните на нем, чтобы выделить его, а затем выберите Format⇨Order⇨Bring to Front (Формат⇨Упорядочить⇨На передний план), чтобы элемент управления оказался поверх других.
- ✓ Если нужный вам элемент управления полностью закрыт другими элементами управления, выделите самый верхний из них и отправьте его под все остальные, выбрав Format⇨Order⇨Send to Back (Формат⇨Упорядочить⇨На задний план). Повторяйте эту операцию до тех пор, пока нужный вам элемент управления не окажется наверху.
- ✓ Если вы специально разместили элементы управления так, чтобы во время выполнения программы они пересекались, используйте команды Format⇨Order⇨Bring Forward (Формат⇨Упорядочить⇨Переместить вперед) и Format⇨Order⇨Bring Backward (Формат⇨Упорядочить⇨Переместить назад), чтобы разместить пересекающиеся элементы управления так, как вам нужно. Эти команды перемещают элемент управления в стопке на одну позицию вперед или назад.

Одновременное форматирование нескольких элементов управления

Многие команды меню **Format** предназначены для работы сразу с несколькими элементами управления или с несколькими группами элементов управления. Все эти команды будут обсуждаться в данном разделе, но сначала давайте поговорим о том, какой из нескольких выделенных элементов оказывается важнее.

Элементы управления и доминирование

Нет, этот раздел не отступление от темы ради развлечения или каприза. Для некоторых команд форматирования, предназначенных для работы сразу с несколькими элементами управле-

ния, один из элементов управления служит отправной точкой для выполняемой команды. На жаргоне VBA такой элемент управления называется *доминирующим элементом управления*.

Когда используется команда Format⇨Make Same Size (Формат⇨Сделать одного размера), например, чтобы получить набор элементов управления одного размера, VBA копирует размеры (высоту, ширину или оба значения) доминирующего элемента управления для остальных выделенных элементов управления. То же самое происходит и для команды Align (Выровнять)— другие элементы управления выравниваются по доминирующему элементу управления, который остается на месте. Результат выполнения команд Horizontal Spacing (Расстояние по горизонтали) и Vertical Spacing (Расстояние по вертикали) из меню Format тоже зависит от того, какой элемент управления доминирует.

На рис. 10.9 видно, что только у одного из выделенных элементов управления маркеры изменения размеров — белые. Это — доминирующий элемент управления, признаком которого как раз и служат белые маркеры. Другие выделенные элементы управления окружены черными маркерами.

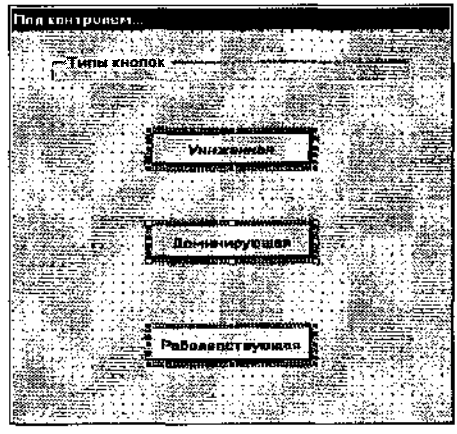


Рис. 10.9. Элемент управления, вокруг которого видны белые маркеры изменения размеров, является доминирующим элементом управления

Вы уже знаете, что выделить группу элементов управления можно двумя способами: охватывая элементы управления прямоугольником выделения с помощью перетаскивания, предварительно выбрав пиктограмму стрелки в панели элементов управления, либо выполняя <Shift+щелчок> или <Ctrl+щелчок> на каждом из элементов управления по одному. Следующая таблица поясняет, как при этом выбрать доминирующий элемент управления.

Стратегия выделения элементов управления	Техника выбора доминирующего элемента
Выбор доминирующего элемента при выделении с помощью перетаскивания выделяющего прямоугольника	Начинать перетаскивание ближе к одному элементу управления, чем к другим
Выбор первого из выделяемых элементов управления в качестве доминирующего	Выделять все элементы управления в группе с помощью комбинации <Shift+щелчок>
Выбор последнего из выделяемых элементов управления в качестве доминирующего	Выделять все элементы управления в группе с помощью комбинации <Ctrl+щелчок>
Выбор другого доминирующего элемента управления в имеющемся наборе выделенных	Дважды выполнить комбинацию <Ctrl+щелчок> на выбранном элементе управления

Выравнивание элементов управления

Даже при включенной сетке довольно часто оказывается, что элементы управления, которые должны были *бы* находиться на одной линии, на самом деле располагаются зигзагообразно. Вместо того чтобы исправлять это вручную, воспользуйтесь командами **Align** (Выровнять), которые сделают все за вас. Эти команды позволяют выставить на одной линии соответствующие стороны всех выделенных элементов управления как по горизонтали, так и по вертикали.

Чтобы выровнять несколько элементов управления, сделайте следующее.

1. **Выделите элементы управления и тот из них, который вы хотели бы оставить на месте, выберите в качестве доминирующего (см. предыдущий раздел).**
2. **Выберите Format⇨Align из меню, а затем конкретный тип выравнивания из появившегося подменю.**

То же самое можно получить, щелкнув на кнопке Align (Выровнять) в панели инструментов UserForm (если эта кнопка представляет нужный тип выравнивания) или выбрав подходящий тип выравнивания из открывающегося меню этой кнопки.

Приведение к одному размеру

Три имеющиеся в меню Format команды Make Same Size (Сделать одного размера) автоматически приведут все выделенные элементы управления к размерам доминирующего элемента управления. Немедленное выравнивание размеров предусмотрено отдельно для ширины и высоты, а также для обеих этих характеристик сразу. Эти три команды присутствуют также в раскрывающемся меню соответствующей кнопки панели инструментов UserForm.

Выравнивание горизонтальных и вертикальных расстояний

С помощью команд Horizontal Spacing (Расстояние по горизонтали) и Vertical Spacing (Расстояние по вертикали) из меню Format можно менять расстояния между выделенными элементами управления, причем четырьмя вариантами на выбор. Из них три варианта оказываются весьма полезными, а один из этих трех доступен только тогда, когда выделено не меньше трех элементов управления. Вот как эти варианты работают.

- ✓ **Make Equal** (Сделать одинаковыми). Выравнивает расстояния между выделенными элементами управления (число последних должно быть не менее трех). Крайние элементы управления остаются на месте, а элементы управления между ними сдвигаются. Если элементов управления только два, команда работать не будет.
- ✓ **Increase** (Увеличить) и **Decrease** (Уменьшить). Увеличивает или уменьшает расстояния между выделенными элементами управления на величину, соответствующую одному шагу сетки в выбранном вами направлении. Доминирующий элемент управления остается на месте, а остальные сдвигаются.
- ✓ **Remove** (Удалить). Сдвигает элементы управления так, что между ними не остается просвета и их стороны касаются одна другой. Доминирующий элемент управления остается на месте.

Элементы управления, tfequMe себя хорошо!

В данном разделе вы ближе познакомитесь с самыми важными элементами управления VBA, а также узнаете, какую роль они играют в программах (в начале настоящей главы вы просто познакомились с ними). Перед тем как рассмотреть каждый элемент управления по

отдельности, я хочу обсудить несколько свойств, относящихся ко многим элементам управления. Некоторые из этих свойств применяются и к формам.

Запомните, что можно проверить работу элементов управления во время написания цикла, не используя для этого никакого программирования. Как только вы запустите форму в окне редактора Visual Basic, элементы управления будут готовы к работе. Например, если вы щелкнете на кнопке, она будет выглядеть нажатой. Элементы управления можно использовать для чего-то полезного. Однако могут возникнуть другие ситуации, о которых я расскажу в разделе "Программирование форм", дальше в настоящей главе.

Использование свойств Enabled и Locked

Два свойства Enabled и Locked управляют тем, будут ли элементы управления или вся форма доступны пользователю. Очевидно, вы захотите получить полный доступ к элементам управления, находящимся в форме. В противном случае возникает вопрос, зачем же они тогда находятся в форме? Однако иногда элементы управления для вас недоступны. Очень часто элементы управления находятся в форме, однако в данный момент они окрашены в серый цвет. Это говорит о том, что использовать сейчас их нельзя. Например, если в текстовом процессоре не выделен какой-нибудь фрагмент текста, команда Cut (Вырезать) не доступна, так как вырезать нечего.

Свойство Enabled определяет, будет ли элемент управления или форма, находящиеся в фокусе, реагировать на движение мыши или нажатие клавиши на клавиатуре. В фокусе Windows может находиться только один объект. Для того чтобы показать, что данный объект находится в фокусе, Windows размещает вокруг данного элемента управления пунктирные границы.

Когда свойство Enabled равно True, элемент управления появляется обычным образом и может находиться в фокусе. Когда свойство Enabled равно False, Windows отображает на экране неактивную (серую) версию этого элемента управления, который не может находиться в фокусе (рис. 10.10).

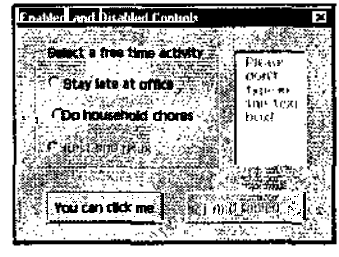


Рис. 10.10. В этом диалоговом окне есть активные и неактивные переключатели и кнопки, а также неактивное текстовое поле

Свойство Locked определяет, что выполняет элемент управления. Если свойство Locked равно True, можно щелкать на элементе управления, использовать любые комбинации клавиш, бросать в него камнями, но элемент управления ничего не будет делать (обратитесь к разделу "Назначение быстрых клавиш" дальше в настоящей главе). Однако если свойство Enabled равно True, элемент управления будет находиться в фокусе и нормально выглядеть.

Настройка порядка перехода по нажатию клавиши табуляции

К наиболее интересным особенностям Windows можно отнести то, что при нажатии клавиши <Tab> фокус перемещается с одного элемента управления на другое. При этом элемен-

ты управления выделяются после нажатия клавиши <Tab>. (Для того чтобы перемещаться в обратном порядке, воспользуйтесь комбинацией клавиш <Shift+Tab>.)

Вы не должны запускать форму для того, чтобы перейти от одного элемента управления к другому. Нажмите клавишу <Tab> в окне UserForm для того, чтобы перейти от одного элемента управления к следующему.

Переход по клавише табуляции основан на порядке, в котором элементы управления добавлялись в форму. Можно добавить элементы управления и затем удалить ненужные.

А что делать для того, чтобы перейти от одного элемента управления к другому? Обычно для этого используются кнопки для перемещения влево и вправо, а также вверх и вниз, однако иногда элементы управления можно пропустить. Проще всего изменить порядок расположения элементов управления, выполнив View⇨Tab Order (Вид⇨Порядок вкладок). На экране появится небольшое диалоговое окно, показанное на рис. 10.11. Для того чтобы переместить элементы управления в списке, находящемся в данном диалоговом окне, щелкните на элементе управления, который необходимо переместить, после чего щелкните на кнопке Move Up (Вверх) или Move Down (Вниз).

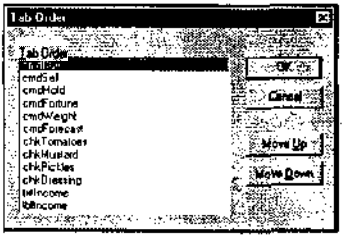


Рис. 10.11. Используйте диалоговое окно Tab Order для того, чтобы контролировать, как пользователи перемещаются по форме VBA с помощью клавиатуры

Порядок размещения элементов управления контролируется с помощью свойства TabIndex. Значение свойства TabIndex равно 0 для первого элемента управления, 1 — для второго элемента управления и т.д. Как только вы измените значение этого свойства, VBA автоматически изменит и другие значения.

Для того чтобы удалить элемент управления из списка, установите значение TabIndex, равное False. Это не изменит расположение элемента управления. Если вы снова установите значение свойства TabIndex, равное True, элемент управления появится там же, где и раньше.

Назначение быстрых клавиш

Несмотря на то, что многим людям достаточно мыши для выбора элементов управления, некоторые предпочитают пользоваться еще и клавиатурой. Для того чтобы упростить работу пользователей, используются быстрые клавиши. После того как форма запущена, нажмите и удерживайте клавишу <Alt>, а потом нажмите определенную клавишу для того, чтобы переместить фокус к определенному элементу управления.

Для того чтобы назначить клавиши, введите один символ в поле Accelerator (Клавиша) в диалоговом окне Properties (Свойства). В заголовке элемента управления должен быть один символ. Также в одной и той же форме не должны указываться одни и те же клавиши для различных элементов управления. VBA автоматически подчеркнет быструю клавишу.

Для того чтобы добавить быструю клавишу для элемента управления, которое не имеет свойства Caption, такого как текстовое поле или полоса прокрутки, выполните следующее.

1. **Создайте метку для элемента управления.**
   О метках я поговорю в следующем разделе.
2. **Настройте порядок расположения таким образом, чтобы метка находилась перед другими элементами управления.**
3. **Назначьте быстрые клавиши** для **метки.**

Теперь после того, как пользователь воспользуется быстрой клавишей, фокус переместится на следующее за меткой элемента управления. На рис. 10.12 показана форма, которая использует эту технику.

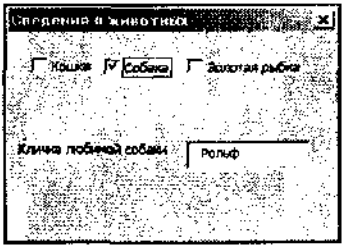


Рис 10.12. Если вы воспользуетесь комбинацией клавиш <Alt+N>, фокус переместится в поле, в котором можно ввести имя своего любимого животного

Отправка сообщений с надписями

Надписи предоставляют прямоугольную область в форме, в которой можно ввести сообщения. С точки зрения пользователя программы, надпись — это не элемент управления; она не позволяет пользователю что-либо контролировать. Надписи представляют собой текст или рисунок. Пользователь не может изменить существующий текст или скопировать его в буфер обмена.

Таким образом, для программиста надписи важны, так как они позволяют создавать сообщения для общения с пользователями. Обычно элементы используются для идентификации элементов управления и их функций. Подобный пример приведен на рис. 10.13. Это особенно полезно для элементов управления без подписей, таких как полоса прокрутки или счетчики.

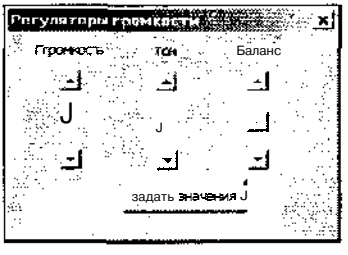


Рис. 10.13. Элементы управления заголовками в верхней части данной формы устанавливают функцию для каждой полосы прокрутки

Элемент может выглядеть как стандартное текстовое поле и не позволяет пользователю копировать содержащийся в нем текст. Установите свойство элемента управления заголовками SpecialEffect равным 2 (для sunken) и свойство BackColor — белый.

Вводэлементовуправлениятекстом

Элементы управления отображают текст, который называется заголовком. Для того чтобы разместить свой собственный текст на элементе управления, измените текст в свойстве Caption в окне Properties (Свойства). Также можно изменить текст прямо в элементе управления, для чего дважды щелкните на нем. После этого рядом с текстом появится курсор. Для того чтобы перейти на новую строку, воспользуйтесь комбинацией клавиш <Shift+Enter>.

Автоматическаянастройкаэлементовуправления

Элементы управления могут автоматически настраивать себя, изменяя содержащийся в них текст. Для этого необходимо изменить параметры в окне Properties (Свойства). На рис. 10.14 показан пример работы этих свойств. Например, можно выполнить следующее.

- ✓ Оставьте значение свойства Wordwrap равным True (по умолчанию), если вы хотите, чтобы VBA автоматически разделял текст на отдельные строки и размещал их в пустом пространстве, как это делает любой текстовый процессор. Если вы установите значение свойства Wordwrap равным False, все заголовки текста останутся в одной строке.
- ✓ Установите значение свойства AutoSize равным True, если вы хотите, чтобы размер элемента управления изменялся автоматически. Если значение свойства WordWrap будет также равно True, элементы управления размещаются вертикально. Если изменить значение свойства Wordwrap на False, элементы управления увеличатся таким образом, что займут одну строку.
- ✓ Используйте свойство TextAlign для того, чтобы контролировать, каким образом текст располагается внутри элемента — слева, в центре или справа.

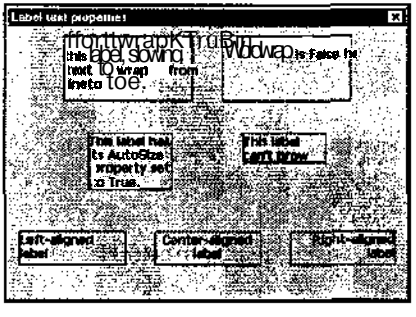


Рис. 10.14. Элементыуправления,содержащиесявданнойформе,демонстрируют различныепараметрысвойств Word-Wrap *и* AutoSize

Автоматическое изменение бывает полезно, однако иногда это приводит к проблемам. Как только элемент управления увеличится, он может стать непропорциональным другим элементам формы. Точнее, элемент может быть слишком велик. Это приводит к тому, что он закроет другие элементы формы или выйдет за границы формы. Если вы используете автоматическое изменение размера, необходимо внимательно следить за тем, чтобы текст на элементах управления никогда не был слишком большим.

Изменениеэлементовуправлениятекстомвкоде

Несмотря на то, что пользователи не могут изменять текст на элементах управления, когда программа запущена, можно изменить значение элемента управления Caption. Вам необходимо изменить всего одну строку в коде для того, чтобы изменить отображаемый текст. Ниже показано, как это сделать:

```
LblInspirationalMessage.Caption = "Laugh and be happy!"
```

Для того чтобы в элементе управления текстом сделать обрыв строки, используйте строковый литерал, связанный с символом возврата каретки в операторе присвоения. (Более подробная информация об объединении текста содержится в главе 11.)

Сбор информации с помощью текстовых полей

Когда необходимо собрать информацию о пользователе, используется текстовое поле. Ваш код может извлекать все, что пользователь введет в данное текстовое поле. На рис. 10.15 показано подобное текстовое поле.

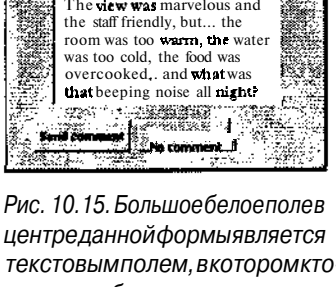


Рис. 10.15. Большоебелоеполев центреданнойформыявляется текстовымполем, вкоторомкто-товвелсообщение

Если вы попытаетесь создать диалоговое окно со всего одним текстовым полем для ввода информации о пользователе, вероятнее всего, вам не нужна вся форма. Вместо этого можно воспользоваться функцией InputBox (подробнее о ней я расскажу в главе 11). Элементы управления текстовым полем можно использовать вместо функции InputBox в следующих случаях.

- ✓ Вам необходимо сделать красивое или отвратительное диалоговое окно. В этом случае функцию InputBox использовать нельзя.
- ✓ Вам необходимо, чтобы в одном диалоговом окне, кроме текстового поля, находился еще один элемент управления.
- ✓ Вы хотите проверять правильность содержимого поля, когда пользователь вводит текст.

Используя текстовое поле, можно сделать программу с процедурой обработки событий, которая будет выполняться всякий раз, как пользователь нажмет любую клавишу на клавиатуре. При этом проверяется, соответствует ли нажатая клавиша установленным вами критериям. Используя функцию InputBox, вы можете только проверять содержимое и только после того, как пользователь закроет текстовое поле.

Размещение стандартного текста в текстовом поле

В текстовом поле можно разместить стандартный текст. При этом пользователю не придется вводить данные в текстовом поле, если содержащаяся в нем информация подходит для него. Для того чтобы ввести в текстовое поле стандартный текст, щелкните один раз в текстовом поле, выделив его, После этого повторно (но не два раза) щелкните в текстовом поле для того, чтобы перейти в режим ввода текста. Затем вводите текст. Также можно ввести текст в поле свойств Value в окне Properties (Свойства). *{Запомните:* текстовые поля не имеют заголовков.)

В действительности текстовое поле имеет как свойство Value, так и свойства Text. Оба эти свойства одинаковы для текстовых полей, однако многие другие элементы управления, такие как кнопки и полосы прокрутки, имеют только свойство Value, но не имеют свойства Text. Так как свойства Value и Text функционально одинаковы для текстовых полей, они равноценны. Я использую свойство Value только потому, что мне легче запомнить его параметры, а также потому, что многие другие элементы управления имеют это же свойство. Для того чтобы обсудить использование свойства Value (или Text) на примерах, ознакомьтесь с разделами "Извлечение информации, введенной пользователем" и "Использование стандартных свойств элемента управления" ниже в настоящей главе.

Извлечение информации, введенной пользователем

Для того чтобы выяснить, какой текст ввел пользователь в текстовом поле, программа должна извлекать свойства Value или Text. Обычно вы должны назначить свойства для строковой переменной с оператором, как показано ниже:

```
strTextBoxText =txtMessageFromUser.Value
```

После запуска программы переменная strTextBoxText теперь содержит любой тип пользователя в текстовом поле, которое называется xtMessageFromUser.

Использование стандартных свойств элемента управления

Многие элементы управления обладают стандартными свойствами, к которым относится свойство Value,

В случае текстового поля вы не должны явно указывать, какое свойство используется — Value или Text — для настройки и извлечения содержимого текстового поля в коде VBA. Поскольку свойство Value является *стандартным свойством* текстового поля, можно его пропустить. Следующая инструкция размещает сообщение в текстовом поле:

```
txtRUListening = "Do as I say!"
```

В данном примере в текстовом поле размешается текст, который пользователь ввел в строковой переменной:

```
strWhatHeard = txtSoundOffTextBox
```

Создание автоматически изменяющихся текстовых полей

Текстовые поля имеют те же свойства AutoSize, Wordwrap и TextAlign, что и элементы управления. Работают они почти так же. Более подробная информация об этих свойствах содержится в разделе "Отправка сообщений с надписями" раньше в настоящей главе. Однако свойство Wordwrap работает только в текстовых полях, состоящих из нескольких строк, о чем я расскажу в главе 19, "Еще о VBA-формах".

Использование кнопок

Если вы хотите что-либо сделать, и сделать именно сейчас, проще всего нажать кнопку и получить немедленный результат. *Кнопки* создают ощущение власти над программой.

Стандартная кнопка представляет собой серый объект, на котором находится поясняющий текст, например OK, Cancel или Guess Again, Friend. Если простой текст оскорбляет ваш изысканный вкус, на кнопку можно поместить небольшое изображение. Для того чтобы сделать это, в окне Properties (Свойства) в поле свойства Picture щелкните на небольшой кнопке, на которой изображены три точки. Вы получите доступ к диалоговому окну, которое позволит указать файл с рисунком. На рис. 10.16 показаны различные кнопки.

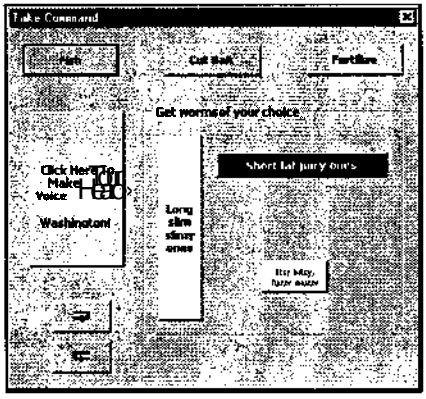


Рис. 10.16. Кнопки могут быть различного размера, формы и формата. На двух кнопках, расположенных в нижнем левом углу, вместо текста находятся рисунки

К сожалению, кнопки ничего не будут делать, если о них ничего не сказано в программе. Щелчком на кнопке вызывается событие Click, однако вы должны написать код для того, чтобы сказать VBA, какое действие должно произойти, когда происходит событие.

Выбор стандартных кнопок

В основной части диалоговых окон нажатие клавиши <Enter> приводит к нажатию кнопки. Эта кнопка является *стандартной кнопкой*, так как именно она реагирует на нажатие клавиши <Enter>, за исключением случая, когда фокус переместился на другой элемент управления.

Для того чтобы назначить кнопку стандартной кнопкой в форме, установите значение свойства Default равным True. Конечно, в форме может быть только одна стандартная кнопка.

Создание кнопки Cancel

Если диалоговое окно позволяет изменить данные или параметры настройки, неплохо было бы предоставить пользователю возможность отмены внесенных изменений. По договоренности, кнопка, предназначенная для отмены сделанных изменения, называется Cancel (Отменить). Если вы не придерживаетесь принятых обозначений, можно назвать эту кнопку •'Never mind" или "Forget it". Независимо от того, что именно написано на данной кнопке, она предназначена для отмены предыдущих операций.

Также по договоренности, после нажатия клавиши <Esc> любое диалоговое окно закрывается, как будто пользователь щелкнул на кнопке, на которой написано Cancel (Отменить) (или что-либо другое). Если свойство кнопки Cancel равно True, это означает, что после нажатия клавиши <Esc> программа будет вести себя так, как будто вы щелкнули на кнопке Cancel (Отменить).

Если свойство Cancel равно True, это автоматически не означает, что после щелчка на кнопке диалоговое окно будет закрыто. Это связывает клавишу <Esc> с событием кнопки Click.

Использование рамок

Рамка — это очень важный элемент форм VBA и представляет собой простой прямоугольник с заголовком в верхней части. В рамке можно разместить другие элементы управления.

Рамки служит для следующих двух целей.

- ✓ Для визуального выделения группы связанных элементов управления. Это помогает понять пользователю, что эти элементы управления связаны. Также рамки позволяют разделить большие формы на несколько участков (рис. 10.17).
- ✓ Для выделения группы кнопок, из которых пользователь должен выбрать только одну.

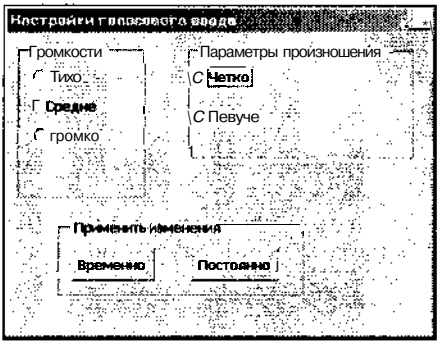


Рис. 10.17. В данной форме используется три различные рамки для упорядочения групп кнопок

В разделе "Выбор элемента с помошью переключателя", дальше в настоящей главе, я подробно рассматриваю использование рамок. В данном разделе я коснусь основ — использования рамок для организации всех типов элементов управления.

Размещение элементов управления в рамках

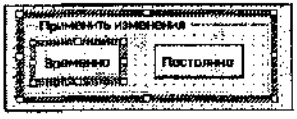
Как только вы добавите рамку в форму, размещение любых других элементов управления в рамке свяжет их и рамку. Теперь при перемещении рамки элементы управления переместятся вместе с ней. При этом они будут занимать те же места в рамке, что и раньше.

Добавить элемент управления в рамку можно следующим образом.

- ✓ Нарисовать новый элемент управления в рамке. Создание нового элемента управления происходит обычным образом. Для этого необходимо щелкнуть на соответствующем значке в панели инструментов и затем перетянуть его на то место в форме, на котором он будет находится. В нашем случае — в рамку.
- ✓ Переместить существующий элемент управления в рамку. Перетягивайте элемент управления с помощью мыши до тех пор, пока указатель мыши не окажется в рамке. Как только вы отпустите кнопку мыши, элемент управления окажется в рамке.

После того как вы разместите элемент управления в рамке, границы рамки станут выделенными, как будто выбран данный элемент управления (рис. 10.18).

Рис. 10.18. После того как вы выберете элемент управления, связанный с рамкой, данная рамка тоже будет выглядеть выбранной



Удаление элементов управления из рамки

Для того чтобы разорвать связь между элементом управления и рамкой, достаточно просто перетянуть данный элемент управления в другое место формы. Как только указатель мыши окажется вне границ рамки, отпустите кнопку мыши. При этом связь между элементом управления и рамкой будет разорвана, а сам элемент управления окажется в другом месте. Теперь оба объекта (форма и элемент управления) можно перемешать отдельно.

Выбор элемента с помощью переключателя

Как в жизни, так и в программном обеспечении, есть много взаимоисключающих вариантов. Покупая мороженое, вы выбираете либо с изюмом, либо с орехами, либо в фруктами, но никогда не все три сразу. Покупая платье или брюки, вы всегда покупаете только те вещи, которые подходят вам по размеру. А когда вы собираетесь жениться на Анне, о других претендентках вы не думаете (Анна должна этому радоваться!).

В Windows для изображения взаимно исключающих вариантов выбора используются так называемые *переключатели*. Это небольшие круглые кнопки, которые работают как нажимные кнопки на радиоприемниках в автомобилях. Одновременно можно слушать только одну радиостанцию. На рис. 10.19 показан типичный набор переключателей.

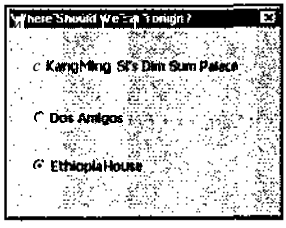


Рис. 10.19. Переключатели в действии

Переключателей всегда должно быть несколько, но выбрать можно только один из них. Если вы выберете один из переключателей, все остальные автоматически останутся не выбранными.

Группировка переключателей

Не беспокойтесь о том, как создать группу переключателей. Все, что вам необходимо сделать, — это разместить кнопки в одном месте в форме. VBA автоматически объединит их в группу. При запуске программы будет выбран только один переключатель.

А что это за странные слова: переключатели должны быть размещены "в одном месте формы"? Конечно, это не официальная терминология VBA, но можно сказать следующее: одна часть формы является самой формой. Каждая добавленная вами рамка элемента управления создаст другую часть формы. И каждая вкладка элемента управления, состоящего из нескольких вкладок, также является отдельной частью формы (о элементах управления, состоящих из нескольких вкладок, которые создаются для ноутбуков и которые похожи на обычные диалоговые окна Windows, я расскажу в главе 19). Можно разместить рамку внутри другой рамки или на вкладке элемента управления, состоящего из нескольких вкладок. Каждая вложенная рамка занимает свою часть формы.

Если в форме есть несколько рамок, VBA рассматривает переключатели, которые не находятся внутри любой рамки, как одну группу, а переключатели, находящиеся в рамке, — как отдельную группу. На рис. 10.20 видно, что я имею в виду.

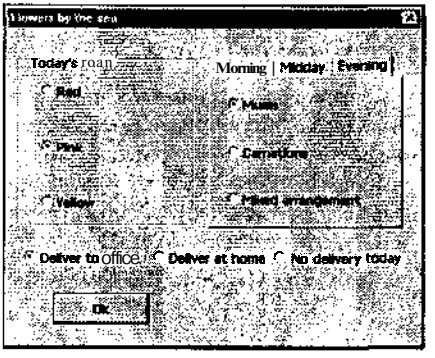


Рис. 10.20. Три отдельные группы переключателей

Как выбрать переключатель

Для того чтобы выбрать переключатель, достаточно просто щелкнуть на нем. Однако обычно после выбора переключателя немедленно ничего не происходит. Диалоговое окно останется открытым. Это позволит пользователю еще раз подумать и, может быть, выбрать другой переключатель. И только после того, как пользователь щелкнет на кнопке OK, он подтвердит свой выбор.

Перед программистом стоит задача: как понять, какой переключатель выбран? Для этого вы должны проверить значение свойства Value для каждого переключателя в группе. Правда, есть способ обойти это. Для этого можно воспользоваться инструкцией If. . .ElseIf:

```
If OptionButton1.Value = True Then
     ChosenOption = "Bill"
Elself OptionButton2.Value = True Then
     ChosenOption = "Bob"
Elself OptionButton3.Value = True Then
     ChosenOption = "Barney"
Else
     ChosenOption = ""
End If
```

Выбор параметров с помощью флажков и кнопок с фиксацией

Переключатели полезны, когда приходится иметь дело с большим количеством взаимоисключающих вариантов. Однако, когда необходимо выбрать несколько вариантов, лучше воспользоваться флажками или кнопками с фиксацией. Флажки и кнопки с фиксацией используются для выбора одного из пары противоположных вариантов, например Yes (Да) или No (Нет). On или Off, True или False и Stay или Leave. На практике отличие между флажком и кнопкой с фиксацией состоит в том, как они выглядят.

✓ *Флажок* — это маленький квадратик, в котором появляется галочка, если выбран параметр Yes, On или True. (Если квадратик пустой, значит флажок *сброшен.)*

✓ *Кнопка с фиксацией* похожа на обычную кнопку. Единственное отличие состоит в том, что, когда вы щелкаете на ней, она остается нажатой.

На рис. 10.21 показаны несколько флажков и кнопок с фиксацией.

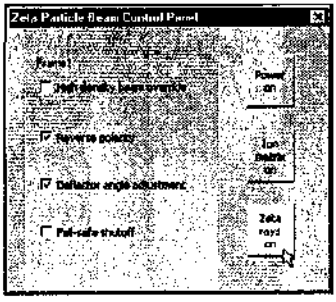


Рис. 10.21. Флажки и кнопки с фиксацией

Группировка флажков

Флажки часто объединяют в группы для того, чтобы составить список не взаимоисключающих вариантов выбора. На рис. 10.22 приведен пример флажков. Обратите внимание на то, что каждый отдельный флажок показывает, выбран ли элемент, возле которого он установлен.

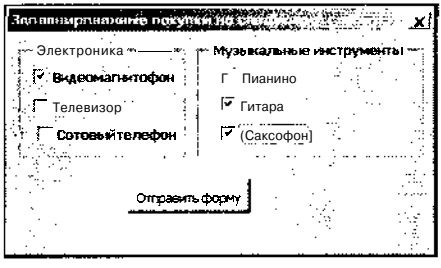


Рис. 10.22. Устанавливать и сбрасывать флажки можно независимо

Установка флажков и выбор кнопок с фиксацией

Как обычно, свойство Value содержит информацию, относящуюся к выбору параметров пользователем. Если флажок установлен, свойство Value равно True, если флажок сброшен, свойство Value равно False. Такая же ситуация и с кнопками с фиксацией: свойство Value равно True; если кнопка нажата. В противном случае свойство Value равно False. Для выбора параметров можно воспользоваться следующим кодом:

```
If tglLightSwitch.Value = True Then
    TurnLightOff
Else
    TurnLightsOff
End If
```

Если вы просто хотите изменить текущее состояние флажка или кнопки с фиксацией, лучше всего воспользоваться оператором Not. В следующем коде флажок устанавливается, если он был сброшен, и сбрасывается, если он был установлен:

```
ChkYesOrNo.Value = Not chkYesOrNo.Value
```

Выбор параметров из списка и комбинированных окон

Если вы предполагаете, что для элемента может быть четыре-пять взаимоисключающих вариантов выбора, созданное вами диалоговое окно будет слишком загроможденным, ведь в нем может быть 10 –12 флажков. Пользователь может устанавливать флажки в зависимости от собственных предпочтений. Но из-за такого количества элементов в форме станет слишком тесно.

Что такое список

Список— это возможность, предоставляемая Windows для решения многих проблем. В списке содержится перечень параметров, которые пользователь может выбрать (рис. 10.23).

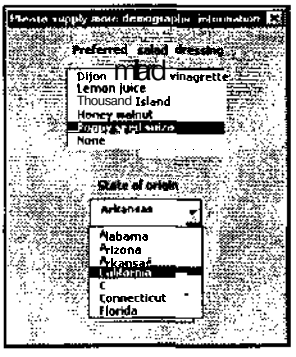


Рис. 10.23. В данной форме в верхней части располагается список, а в нижней - комбинированное окно

С помощью списка нельзя получить доступ к элементам, которые не содержатся в нем. Кроме того, нельзя представить список VBA в виде раскрывающегося списка в одной строке. Для того чтобы избежать этого ограничения, необходимо воспользоваться комбинированным списком.

Комбинированные списки

Комбинированный список объединяет в себе достоинства простого списка и текстового поля. Пользователь может выбрать необходимый элемент из списка, а если такового не окажется в списке, ввести его. Основное отличие между списком и комбинированным списком заключается в следующем: для того чтобы раскрыть весь комбинированный список, необходимо щелкнуть на кнопке, на которой изображен треугольник. Скорее всего, вы знакомы с комбинированными списками, но посмотрите на рис. 10.24, чтобы вспомнить, как он выглядит.

С точки зрения пользователя комбинированные списки намного лучше, так как пользователь может легко найти в них необходимый элемент. Однако программисты во многих ситуациях предпочитают офаничивать выбор пользователя, чтобы предотвратить ввод неправильных данных.

Узнав, что такое список, не пользуйтесь им

Я советую использовать комбинированные списки для контроля всех параметров, которые содержатся в списке, независимо от того, вводит или нет пользователь текст, которого нет в данном списке. Забудьте о простых списках.

И вот почему: простой список VBA не может отображать элементы в виде раскрывающегося списка, Вместо этого вы имеете дело с списком, в котором перечислены все доступные варианты выбора сразу. Это не всегда удобно, так как, если в списке слишком много элементов, он займет слишком большую часть формы. Если элементов в списке немного, лучше воспользоваться переключателями или флажками.

В то же время комбинированные списки более компактные, так как они всегда занимают только одну строку. Для того чтобы комбинированный список превратился в обычный простой список, достаточно установить значение свойства `Style` равным 2 (`fmStyleDropDownList`).А теперь подумайте: стоит ли мучиться со списками?

Размещение элементов в списке или комбинированном списке

Теперь пришла очередь самого трудного. Окно Properties (Свойства) нельзя использовать для ввода вариантов выбора, которые должны находиться в списке или комбинированном списке. Вместо этого вы должны написать код для метода `AddItem` элемента управления или связать элемент управления с *источником данных* (он представляет собой электронную таблицу Excel или базу данных Access).

Для того чтобы создать список прямо в коде, используется соответствующая процедура для события `Activate` данной формы. Она должна содержать несколько инструкций, которые похожи на приведенный ниже пример:

```
Private Sub UserForm_Activate()
    cmbOptionPoll.AddItem "Overpopulation"
    cmbOptionPoll.AddItem "Global warming"
    cmbOptionPoll.AddItem "No time to smell the roses"
    cmbOptionPoll.AddItem "No roses to smell"
    cmbOptionPoll.AddItem "Taxes on the rich too high"
    cmbOptionPoll.AddItem "Too many social services"
    cmbOptionPoll.AddItem "Inadequate social services"
    cmbOptionPoll.AddItem "HMOs"
End Sub
```

Технические подробности размещения элементов из базы данных в списке или комбинированном списке выходят за пределы данной книги, однако вы должны знать, что это возможно.

ЧТО же пользователю делать

Для того чтобы найти элемент, который пользователь выбрал или ввел в списке или комбинированном списке, используйте свойство объекта `Value` в коде. Назначьте свойству соответствующую переменную, как показано ниже:

```
strOption = cmbOptionPoll.Value
```

Программирование форм

Добавлять элементы управления в формы несложно, а вот, чтобы заставить их делать то, что вам нужно, потребуется немного больше умственной работы и профаммирования. В этом разделе мы обсудим тонкости процесса профаммирования форм.

Покажите то, что имеете!

Решив добавить в свою VBA-программу пользовательские формы, вы неизбежно столкнетесь с необходимостью показать эти формы на экране. Иначе зачем эти, пусть даже самые лучшие в мире, формы нужны, если их никто не увидит?

Из-за того, что VBA-программа может при необходимости пользоваться интерфейсом содержащего ее приложения, программа не отображает никакую из ее форм автоматически. В этом отношении VBA отличается от своего собрата Visual Basic, где программа по сути *является* формой. Так или иначе, чтобы отобразить форму и сделать ее доступной пользователю, в VBA-программе вам придется добавить программный код.

Загрузка и отображение форм

Процесс отображения формы в VBA состоит из двух шагов;

- ✓ загрузка формы в память;
- ✓ отображение формы на экране.

Оба этих шага можно осуществить с помощью одного оператора VBA, Но иногда полезно разделить эти шаги и использовать для них отдельные операторы.

Отображение окон

Отображение форм осуществляется методом Show. Например, если именем формы является `FormICa`, то нужно просто напечатать:

```
FormICa.Show
```

Заметьте, что Show — это метод объекта `UserForm`, поэтому метод добавляется к имени формы после точки-разделителя. Если указанная форма еще не загружена в память, метод Show сначала загрузит форму, а потом сделает ее видимой,

Загрузка форм без отображения их на экране

Чтобы загрузить форму в память перед отображением, используйте оператор Load. `Load` — это не метод, поэтому в данном случае синтаксис другой (по сравнению с использованием Show):

```
Load ФормАльДеГид
```

Зачем загружать форму, не отображая ее? Загрузка формы происходит намного дольше, чем ее отображение. Программа любой сложности выполняет немало процедур инициализации (типа чтения данных из файлов, вычисления начальных значений переменных и создания объектов), поэтому при запуске программы некоторая задержка весьма привычна. Если в это время вы загрузите и свои формы, пользователь отнесется к задержке благосклоннее, чем к такой же задержке во время работы программы. Единственный недостаток предварительной загрузки форм заключается в том, что для них нужна память, которая могла бы использоваться для других целей.

Внесение изменений в форму перед ее отображением

Можно загрузить форму и без оператора Load, выполнив оператор, в котором вызывается свойство либо метод формы или любого из помещенных в нее элементов управления. При этом можно вносить в форму изменения перед тем, как она отображается на экране, Хотя ок-

но свойств редактора Visual Basic позволяет безо всякого программирования управлять тем, как форма будет выглядеть и вести себя на экране, здесь нужно отметить следующее: часто вообще нельзя узнать, как должна выглядеть и что должна делать форма (или ее элементы управления), пока программа не начнет выполняться.

Предположим, например, что в заголовке формы должны отображаться текущие дата и время. Вы не можете предугадать, когда именно пользователь запустит вашу программу, поэтому вам придется поручить VBA указать дату и время за вас. Это можно сделать, например, так:

```
Sub DisplayDateCaptionedForm()
    DateCaptionedForm.Caption = Now
    DateCaptionedForm.Show
End Sub
```

Аналогично можно заставить надпись или текстовое поле формы отображать информацию о том, что в данный момент выделено в VBA-приложении. На рис. 10.24 показано то, что можно увидеть в результате выполнения следующей процедуры VBA, связанной с подходящей формой в Visio:

```
Sub DisplayShowSelectionForm()
    Dim ItemCount As Integer, Message As String
    Items = ActiveWindow.Selection.Count
    Message = "Выделено объектов: " & CStr(Items) S "."
    ShowSelectionForm.lblCountOfItems.Caption = Message
    ShowSelectionForm.Show
End Sub
```

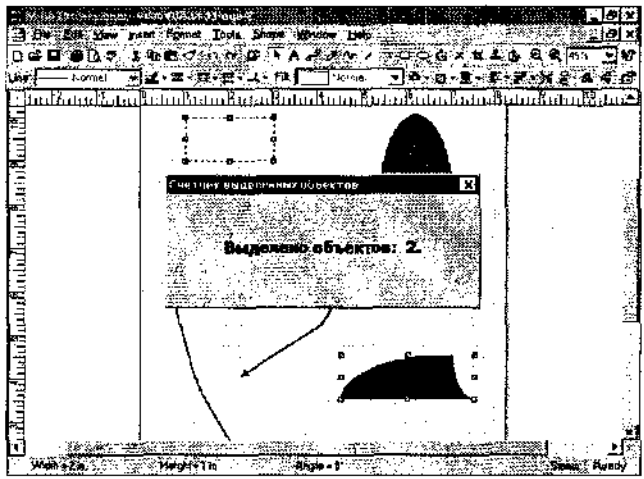


Рис. 10.24. В этой программе содержание текста надписи выясняется непосредственно перед отображением формы

Обратите внимание на строку программного кода, в которой изменяется свойство элемента управления "надпись" в форме:

```
ShowSelectionForm.lblCountOfItems.Caption = Message
```

Технически невозможно получить доступ к свойству или методу формы до тех пор, пока форма не будет загружена, Однако вам не нужно писать для этого явный оператор. VBA автоматически загружает форму, как только потребуется выполнить оператор, ссылающийся на одно из ее свойств или на один из ее методов. Ссылка на свойство или метод любого из элементов управления в форме порождает тот же эффект.

Изменение формы с помощью событий Initialize и Activate

Пример из предыдущего раздела демонстрирует один из способов внесения изменений в форму перед ее отображением на экране — из стандартной процедуры обычного модуля. Но, как говорится, содрать с капусты листья можно многими способами. В нашем случае есть варианты — события `Initialize` (Инициализация) и `Activate` (Активизация) формы. Программный код, помещенный в процедуры обработки этих событий, выполняется автоматически, когда случаются соответствующие события. Выбрать между этими событиями помогут следующие правила.

- ✓ Используйте событие `Initialize` для программного кода, который должен выполняться только при первой загрузке формы.
- ✓ Используйте событие `Activate` для программного кода, который должен выполняться при каждом отображении формы на экране (включая и первое отображение).

Создание программного кода для процедур обработки событий обсуждается ниже, в разделе "Главные события".

Как скрыть видимую форму

Используйте метод `Hide` (Скрыть), чтобы закрыть форму и таким образом получить возможность вернуться в документ VBA-приложения или активизировать другую форму. Это делается следующим оператором:

```
FormErly.Hide
```

Правда, метод `Hide` можно использовать не в любом месте программы. Если форма модальная, метод `Hide` нужно поместить в процедуру обработки события, принадлежащую самой форме. Как упоминалось раньше в настоящей главе, если открыта модальная форма, выполняться могут только процедуры, связанные с этой формой.

И кстати, при вызове метода `Hide` в процедуре обработки события, принадлежащей этой форме, имя формы указывать не требуется — VBA достаточно сообразителен, чтобы понять, что в данном случае метод `Hide` принадлежит именно этой форме. Поэтому в процедуре обработки события, принадлежащей форме, можно просто напечатать

```
Hide
```

и эта форма исчезнет.

Чаще всего оператор `Hide` размещают в конце процедуры обработки события `Click` (Щелчок) для КНОПОК OK, **Отмена** или **Закрыть.** Примеры вы найдете ниже, в разделах "Добавление кнопок **Закрыть** и **Отмена**" и "Программирование кнопки OK".

Скрытая форма не удаляется из памяти, вы сможете отобразить ее снова без лишних задержек с помощью метода `Show` в любое время.

Удаление формы из памяти

Если известно, что форма больше в программе не понадобится, уничтожьте ее совсем, удалив из памяти. В небольших VBA-программах в этом нет необходимости, но когда ваши программы станут большими и память будет как награда, уничтожение ненужных форм приобретет очевидный смысл.

Как и для загрузки формы, для выгрузки формы из памяти используется не метод, а оператор `Unload FormAtion`.

Выгрузка формы удаляет ее и с экрана, если до этого форма была видима.

Если известно, что форма больше в программе не понадобится, замените метод `Hide` в процедурах обработки событий формы на оператор `Unload`. В этом случае оператор `Unload Me` эквивалентен оператору `Unload имя_формы`, поскольку ключевое слово `Me` представляет текущую форму.

Однако, в отличие от метода `Hide`, оператор `Unload` применим и к модальным формам, которые не присутствуют на экране. С помощью оператора `Unload`, размещенного в главной части программы (т.е. в процедуре, не связанной с формами), можно выгружать любые формы, в том числе и скрытые модальные формы.

Главные события

Когда выполняется процедура типа `Sub`, не отображающая форм, ваш программный код полностью контролирует, что и когда делает программа. Но если на экране отображается форма, программа переходит в каком-то смысле в пассивное состояние, ожидая инструкций от пользователя. После каждого нажатия клавиши, перемещения мыши или щелчка на кнопке генерируется программное событие. Ваша программа регистрирует каждое такое событие и проверяет, не содержит ли программный код формы процедуру, связанную с этим событием. Если подходящей процедуры нет, событие пройдет для программы бесследно. Но если форма имеет процедуру, соответствующую событию, то программа сразу же оживет и честно выполнит эту процедуру.

Процедура обработки события может делать все, что и любая другая процедура: вычислять значения переменных, манипулировать свойствами и методами объектов, загружать и отображать другие формы. После завершения выполнения процедуры обработки события контроль возвращается к форме. Программа перейдет в состояние ожидания следующего события.

Типичные события

Формы VBA и элементы управления в них могут распознавать самые различные события. (На жаргоне VBA, когда говорится, что объект "имеет" события, подразумевается, что объект может регистрировать и распознавать эти события.) У форм и элементов управления есть одинаковые события, но набор событий каждого объекта индивидуален. Часто встречающиеся события приведены в табл. 10.1.

Таблица 10.1. Избранные события форм и элементов управления

Событие	Объекты, которые распознают событие	Когда происходит событие
`Activate`	Формы	Каждый раз, когда форма активизируется (получает фокус ввода)
`àddControl`	Формы, фреймы и формы с множеством страниц	При добавлении в объект элемента управления во время выполнения формы

Окончание табл. 10.1

Событие	Объекты, которые распознают событие	Когда происходит событие
AfterUpdate	Все "действующие" элементы управления, кроме кнопок	После установки нового значения для элемента управления, в момент перехода от данного элемента управления к другому
Change	Все "действующие" элементы управления, кроме кнопок	При изменении значения свойства Value элемента управления
Click	Формы и все типы элементов управления	После щелчка кнопкой мыши на объекте
DblClick	Формы и все типы элементов управления	После двойного щелчка кнопкой мыши на объекте
DropButtonClick	Текстовые поля и поля со списком	При появлении раскрывающегося списка (после щелчка на кнопке раскрытия или нажатия клавиши <F4>)
Enter	Все типы элементов управления	Непосредственно перед тем, как элемент управления получит фокус ввода от другого элемента управления в той же форме
Error	Формы и все типы элементов управления	Когда возникает ошибка, но информация о ней не может возвратиться программе
Exit	Все типы элементов управления	Непосредственно перед тем, как фокус ввода перейдет от данного элемента управления к другому в той же форме
KeyUp, KeyDown, Keypress	Формы и все типы элементов управления	При нажатии или отпускании кнопки
Layout	Формы, фреймы и формы с множеством страниц	При изменении размеров объекта
RemoveControl	Формы, фреймы и формы с множеством страниц	При удалении элемента управления из объекта во время выполнения формы
Scroll	Формы, фреймы и формы с множеством страниц, а также текстовые поля, списки и поля со списком	При изменении положения бегунка полосы прокрутки
Zoom	Формы, фреймы и формы с множеством страниц	При изменении масштаба объекта (значения свойства Zoom)

Программирование обработки событий

Как видно из табл. 10.1, формы и элементы управления *могут* отвечать на многие события. Но когда форма действительно отвечает на какое-нибудь конкретное событие? Только тогда, когда она имеет соответствующую процедуру для этого события. Процедуры обработки событий должны быть кем-то созданы — в данном случае вами.

Создание и редактирование процедур обработки событий

Создание процедуры обработки события не отличается от создания любой другой процедуры в VBA. Нужно только знать, куда поместить соответствующие операторы. Программный код для процедуры обработки события, как и весь остальной связанный с формой программный кол, в редакторе Visual Basic размещается в окне программного кода этой формы. Следовательно, процедуры обработки событий для всех элементов управления в форме, как и для самой формы, создаются в окне программного кода формы. Поэтому, перед тем как программировать соответствующие событиям процедуры, выполните следующие шаги.

1. **Откройте окно программного кода формы.**
   Двойной щелчок на форме или любом ее элементе управления — самый короткий путь для этого. Иначе можно выделить форму и выбрать View⇨Code из контекстного (вызываемого щелчком правой кнопки мыши) меню для формы или элемента управления, с которыми вы собираетесь работать, или же нажать <F7>, когда форма выделена в ее окне UserForm.
2. **В окне программного кода формы выберите объект,** для **которого вы хотите создать процедуру обработки события.**
   Объект выбирается из раскрывающегося списка объектов вверху слева в окне программного кода формы (рис. 10.25).
3. **Выберите событие,** для **которого нужно создать программный код.**
   На этот раз используйте раскрывающийся список процедур, размещенный в окне программного кода вверху справа.

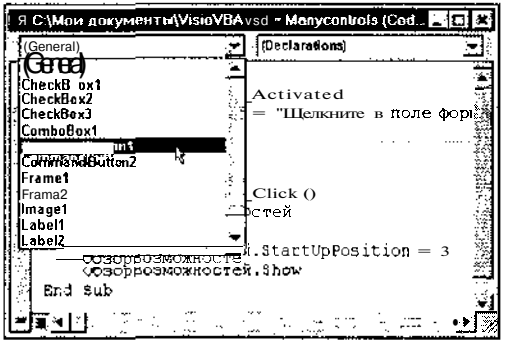


Рис. 10.25. Выбор элемента управления в окне программного кода формы перед началом создания процедуры обработки события

Как только из раскрывающегося списка процедур будет выбрано событие, VBA немедленно перенесет вас прямо к процедуре его обработки. Если для события еще не создано никакого программного кода, VBA создаст для вас заготовку процедуры, разместив курсор ввода в пустой строке между оператором объявления процедуры и ее завершающим оператором (см. рис. 10.25). Если процедура обработки события уже содержит программный код, VBA просто поместит курсор ввода в его первую строку.

Синтаксис процедур обработки событий

Вам не стоит беспокоиться об объявлении процедур, так как VBA делает это автоматически, когда вы выбираете событие в окне программного кода

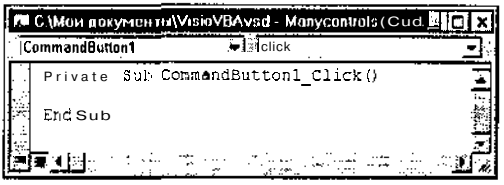


Рис. 10.26. Только что созданная заготовка процедуры обработки события в окне редактора Visual Basic

Как видно из рис. 10.26, синтаксис процедур обработки событий не отличается от синтаксиса обычных процедур типа Sub. Единственное, что отличает процедуру обработки события, так это ее имя. Чтобы процедура обработки события вообще функционировала, ее имя должно состоять из имени объекта (формы или элемента управления), за которым следуют символ подчеркивания и официальное имя события VBA. Посмотрите на следующие примеры:

```
Private Sub cmdCalculateSquareRoot_Click()
End Sub

Private Sub UserForm_Activate()
End Sub

Private Sub sclVolumeControl_Change()
End Sub
```

Когда нужно изменить имя процедуры обработки события

Единственный случай, когда возникают проблемы с именем процедуры обработки события, — когда имя объекта, которому соответствует процедура, изменяется после создания процедуры. VBA не меняет при этом имя процедуры автоматически, поэтому вы должны открыть окно программного кода и привести имя процедуры обработки события в соответствие с новым именем элемента управления.

Например, предположим, что вы создали процедуру обработки события Click (Щелчок) для кнопки, которой было автоматически присвоено имя CommandButton1. Тогда именем процедуры будет CommandButton1_Click. Спохватившись, вы изменили имя кнопки на более информативное, например cmdВернутьИзКорзины. Но после этого, до тех пор пока вы не измените оригинальное имя процедуры на cmdВернутьИзКорзины_Click, соответствующая кнопка во время выполнения будет просто торчать в форме тихо и безответно, независимо от того, сколько раз вы на ней щелкнете.

Поскольку при изменении имени кнопки VBA не удаляет соответствующие этой кнопке процедуры, вам не нужно создавать процедуру вновь, а нужно только изменить ее имя. Можно сделать и по-другому: начать новую процедуру обработки события для кнопки, а затем с помощью команд Cut (Вырезать) и Paste (Вставить) перенести программный код из оригинальной процедуры в новую.

Щелкните здесь...

В Windows щелчок (Click) является квинтэссенцией всех событий. Вам приходится щелкать на пиктограммах, чтобы выбрать их, щелкать в документах — чтобы позиционировать точку ввода, в меню — чтобы открыть их, и на кнопках — чтобы активизировать связанные с ними функции. А раз щелчки столь часто используются в пользовательском интерфейсе Windows, вы

непременно захотите, чтобы созданные вами формы отвечали на щелчки мыши. Однако для многих элементов управления не требуется создавать программный код специально для того, чтобы они отвечали на щелчок (почему это так, объясняется ниже, в разделе "Когда не нужно создавать процедуры обработки события Click"). Но для самого важного изо всех элементов управления — для кнопки — программный код создавать придется.

Очевидно, что каждая кнопка должна иметь процедуру обработки события `Click`, если вы хотите, чтобы кнопка делала что-нибудь полезное после щелчка на ней. Следующая процедура обработки события просто считает и отображает на экране число щелчков на кнопке:

```
Private Sub cmdCountClicks_Click()
' Объявление переменной intCount статической сохранит
' ее значение в промежутках между вызовами процедуры
Static intCount As Integer
intCount = intCount + 1
cmdCountClicks.Caption = "Вы щелкали на этой кнопке " _
     & intCount & " раз(а)."
End Sub
```

Представленный здесь программный код достаточно ясен. При каждом выполнении процедуры, которое происходит только после щелчка пользователя на кнопке, знамение переменной `intCount` увеличивается на 3. Это значение используется в строке, которая отображается на кнопке с помощью свойства `Caption` кнопки. Кстати, объявление переменной `intCount` как статической заставляет VBA сохранять значение этой переменной между вызовами этой процедуры обработки события. Если объявить переменную с помощью ключевого слова Dim, переменная будет инициализироваться при каждом вызове процедуры. Вот процедура обработки события для кнопки с именем `cmdMoveThisForm`:

```
Private Sub cmdMoveThisForm_Click()
Move(Left -24), (Top -24)
End Sub
```

Если вам это интересно, после щелчка пользователя на кнопке эта процедура смещает форму на 24 единицы вверх и на 24 единицы влево. В процедуре всего одна строка выполняемого программного кода, где метод Move используется без прямой ссылки на объект, — VBA предполагает, что ссылка указывает на главный объект формы, т.е. на саму форму. Если нужно вместо формы переместить содержащуюся в ней кнопку, используйте `cmdMoveThisForm.Move`, чтобы указать этот объект как целевой. Точно так же, если нет явной ссылки на объект, то подразумевается, что свойства `Left` и `Top` относятся к форме.

Эти два примера довольно тривиальны, но в то же время они хорошо иллюстрируют, что процедуры обработки событий выглядят и работают точно так же, как и любые другие процедуры.

Превращение закрытых процедур в открытые

Процедуры обработки событий особые потому, что VBA выполняет их автоматически при наступлении определенного события. Но поскольку в остальном процедура обработки события ничем не отличается от обычной процедуры, вы можете вызывать ее из программного кода как обычную. Просто выделив имя процедуры в отдельный оператор, например:

```
cmdПоказФокусов_Click
```

; вы даете указание выполнить процедуру, по сути, заставляя VBA думать, что случилось соответствующее событие.

• По умолчанию процедуры обработки событий локальные (`Private`), т.е. при создании процедуры VBA автоматически добавляет в начало ее объявления ключевое слово `Private`. В таком случае выполнять эти процедуры можно только из программного кода, связанного с данной формой.

Однако ничто не мешает вам удалить ключевое слово `Private` и напечатать вместо него `Public`. Тогда вы сможете вызывать соответствующую процедуру обработки события и из любой другой части программы. Здесь кроется одна тонкость: прежде чем вызывать открытые (`Public`) процедуры формы, форму нужно загрузить в память (но не обязательно показывать на экране).

Когда не нужно создавать процедуру обработки события Click

Большинство элементов управления VBA распознают событие `Click`. Но, за исключением кнопок, обычно нет необходимости и даже неразумно создавать процедуру обработки события `Click`, даже если нужно, чтобы объект отвечал на щелчки кнопки мыши. Причина в том, что эти элементы управления отвечают на щелчки автоматически и обычно так, как требуется.

Предположим, вы поместили в форму VBA несколько кнопок. После запуска формы на выполнение щелчок на кнопке выделяет эту кнопку и для этого не требуется никакого программирования. Точно так же VBA автоматически обрабатывает щелчки на кнопках выключателей и переключателей. VBA знает, что щелчок должен изменить состояние элемента управления на противоположное, если для элемента управления предусмотрено только два состояния (состояние включен/выключен — для выключателя, отмечен/не отмечен — для флажка и т.д.). А после щелчка в поле текста этот элемент управления автоматически перемещает курсор ввода в то место, где вы щелкнули.

Но VBA не может передать информацию о состоянии элемента управления вашей программе — это придется делать вам, создавая программный кол. Что введет пользователь в поле текста? Какой флажок отметит и для какого снимет отметку? Какую из кнопок переключателей выберет? Ваша программа не узнает об этом до тех пор, пока вы не создадите программный код, позволяющий выяснить и обработать эту информацию.

Здесь нужно использовать свойство `Value`, в котором содержится информация о текущем состоянии элемента управления, если этот элемент управления может быть в разных состояниях или содержать данные. Вы можете заставить программу считывать значение свойства `Value` при каждом его изменении, если создадите процедуру обработки события `Change`. Можно также просто прочитать значение свойства `Value` после того, как форма будет скрыта с экрана. Подробнее об этом — в разделе, посвященном событию `Change`, и разделе "Основные приемы программирования форм".

События Click в формах

Сама форма тоже имеет событие `Click`. На практике, конечно, большинство форм используется просто как подложка для размещения кнопок, текстовых полей и других элементов управления. Но при желании вся форма может играть роль одной большой кнопки, и тогда щелчок в любом ее месте будет инициировать выполнение чего-нибудь очень полезного в программе.

Для этого нужно поместить программный код соответствующих действий в процедуру обработки события `Click` формы:

```
Private Sub UserForm_Click()
...(программный код, инициируемый событием)
End Sub
```

Обратите внимание на то, что имена всех процедур обработки событий формы начинаются с UserForm, независимо от того, какое имя вы назначили самой форме. Если вы действительно собираетесь создавать процедуру обработки события Click формы, то должны четко представлять себе, что VBA выполнит эту процедуру после шелчка пользователя только в той части формы, которая не закрыта каким-нибудь элементом управления. Другие процедуры обработки событий, связанных с манипуляциями мышью, работают аналогично.

Создание программного кода для события Change

Для элементов управления типа полос прокрутки, кнопок прокрутки переключателей и выключателей главным событием является событие Change. Эти элементы управления реагируют на щелчки мыши и нажатия клавиш, но реагируют автоматически. VBA вместе с Windows делают всю "грязную" работу по изменению внешнего вида элемента управления и изменению его установок в соответствии с тем, какие клавиши или какие кнопки мыши были нажаты.

Возможно, вы захотите, чтобы программа отвечала не на сами нажатия, а на те изменения, к которым они привели. В таком случае вы должны создать программный код для события Change. Событие Change происходит, когда изменяется значение элемента управления (т.е. когда изменяется значение свойства Value). При этом не имеет значения, почему произошло изменение значения — после шелчка кнопки мыши или печатания либо потому, что какая-то процедура в программе в одном из своих операторов манипулировала значением свойства Value этого элемента управления.

Не исключено также, что вы захотите, чтобы программа отвечала на изменение значения некоторого элемента управления по окончании работы с ним пользователя. В таком случае нужно создать программный код для события AfterUpdate, которое возникает при передаче фокуса ввода другому элементу управления.

С помощью процедуры обработки события можно сделать гораздо больше, чем просто отобразить новое значение элемента управления. Например:

- ✓ проверить, удовлетворяет ли это значение заданным критериям;
- ✓ сначала выполнить вычисления, основанные на этом значении, а затем определенные действия, в зависимости от результатов вычисления;
- ✓ использовать значение элемента управления для того, чтобы установить другое значение, например для громкости звука динамика вашего компьютера.

Распознавание нажатий клавиш

Используйте события Keypress, KeyDown и KeyUp, чтобы отвечать на нажатия клавиш пользователем. Событие Keypress удобно использовать для распознавания клавиш с обычными "печатаемыми" символами (буквы, числа, знаки пунктуации), когда нужно обработать информацию, вводимую в текстовое поле или в поле со списком. С помощью этого события распознаются также многие из комбинаций типа <Ctrl+клавиша>, а также клавиша <Backspace>. Немного позже я покажу, как проверить или изменить напечатанный символ с помощью процедуры обработки события KeyPress.

События KeyDown и KeyUp, напротив, распознают практически любую посылаемую им комбинацию клавиш, включая выкрутасы типа <Alt+Shift+Ctrl+F9>. С этими событиями работать труднее, чем с KeyPress, но зато они позволяют использовать более широкий набор сочетаний клавиш. Например, можно создать процедуру обработки события KeyDown, которая позволяет с помощью комбинаций <Ctrl+←> и <Ctrl+→> соответственно уменьшать или увеличивать значение полосы прокрутки, например, на 10.

Основные приемы программирования форм

Теперь, уяснив, как работают процедуры обработки событий, вы сможете применить свои знания при создании диалоговых окон, которые будут работать так, как хотите вы, и так, как ожидают пользователи. В этом разделе мы обсудим целый ряд сценариев, применяемых при создании диалоговых окон.

Добавление кнопок Закрыть и Отмена

Большинство диалоговых окон имеют, как минимум, одну кнопку главной команды и кнопку, убирающую диалоговое окно с экрана. В зависимости от того, что делает диалоговое окно, на этой кнопке обычно написано либо Закрыть, либо **Отмена,** но вполне допустимы и другие надписи — **Выход, Готово, Пропустить, Закончить** и т.п. Такая кнопка потребуется любой вашей форме.

По общепринятому соглашению кнопки **Закрыть** и **Отмена** просто прячут или выгружают форму, и ничего больше. Следующие рекомендации помогут определить, какая из этих кнопок лучше подойдет для вашей формы.

- ✓ Используйте кнопку **Закрыть** для форм, которые только отображают информацию или выполняют свою задачу немедленно, без изменения установок программы или модификации переменных, которые будут использоваться в программе позже.
- ✓ Используйте кнопку **Отмена** для форм, которые изменяют переменные или установки программы. После щелчка пользователя на кнопке **Отмена** диалоговое окно должно закрыться без записи таких изменений — все должно остаться так, как было до открытия диалогового окна. Такая форма должна также иметь кнопку OK, щелчок на которой подтверждает изменения и вносит их.

Простые процедуры обработки событий для кнопок Закрыть и Отмена

Подобно любым другим кнопкам, для того чтобы отвечать на щелчки на них, кнопки **Закрыть** и **Отмена** требуют создания процедур обработки события Click. В большинстве случаев эти процедуры содержат всего один оператор, как в следующих двух примерах:

```
Private Sub crndClose_C±ick (}
     Hide ' здесь подразумевается ссылка на форму
End Sut

Private Sub cmdCancel_Click()
     Unload frmOptions
End Sut
```

Для любой кнопки можно использовать метод Hide или оператор Unload. Разница между этими двумя способами закрытия формы обсуждалась в разделах "Как скрыть видимую форму" и "Удаление формы из памяти".

Конечно, процедуры, связанные с кнопками **Закрыть** и **Отмена,** могут до закрытия формы сделать что-то еще, например отобразить диалоговое окно, в котором пользователю предлагают подтвердить (или не подтвердить) желание закрыть форму.

```
Private Sub cmdClose_Click()
     Message = "Вы действительно хотите закрыть " _
          & "диалоговое окно и отменить все " _
          & "внесенные вами изменения?"
     If MsgBox(Message, vbYesNo) = vbYes Then
          Hide ' закрыть, если пользователь ответил Да
     End If ' иначе не делать ничего
End Sut
```

Клавишная альтернатива

Не забудьте связать кнопку Закрыть и Отмена с клавишей <Esc>. Пользователи привыкли нажимать <Esc> для выхода из диалоговых окон, и вы не должны обманывать их ожиданий. Тем более, что в данном случае не придется создавать процедуру обработки события KeyPress — просто установите для свойства Cancel значение True в окне свойств.

Программирование кнопки OK

Представьте себе типичное диалоговое окно. Вы ожидаете, что после щелчка на кнопке OK программа воспримет введенные в окно данные как окончательные и соответствующим образом изменит внешний вид, поведение или данные программы. После этого форма должна удалить себя с экрана.

В соответствии с этими ожиданиями вы должны создать и процедуру обработки события Click для кнопки OK. Все, что должен сделать в данном случае программный код,— это передать значения от элементов управления переменным программы или использовать значения элементов управления в условных выражениях. Последняя строка процедуры должна содержать либо метод Hide, либо оператор Unload. В следующих примерах txtCName и txtCAddress представляют текстовые поля. Первых два оператора передают их значения соответствующим переменным программы. Затем программа проверяет состояние выключателя tglSend и, если он включен, выполняет процедуру SendBillToCustomer. Наконец, она прячет форму:

```
Private Sub cmdOK_Click()
     strCustomerName = txtCName.Value
     strCustomerAddress = txtCAddress.Value
     ' проверка состояния выключателя
     If tglSend.Value = True Then
          SendBillToCustomer
     End If
     Hide
End Sub
```

Проверка правильности вводимых данных

Чаще всего в процедурах обработки событий приходится заниматься проверкой данных, введенных пользователем с помощью элементов управления, Обычно программа предполагает ввод данных определенного вида. Однако пользователь может ввести любой текст в текстовое поле или поле со списком и выбрать почти любое число с помощью кнопок прокрутки.

В таком случае в процедуру соответствующего элемента управления крайне желательно добавить программный код, проверяющий правильность ввода. Этот программный код должен введенные пользователем данные оценить с точки зрения удовлетворения заданным критериям. Если критерии удовлетворены, программный код сохраняет введенные значения или передает их другой части программы. В противном случае можно отобразить сообщение или как-то по-другому информировать пользователя о том, что возникла проблема (рис. 10.27). Можно также с помощью этого программного кода конвертировать введенные данные в данные подходящего вида, например сделать все буквы прописными.

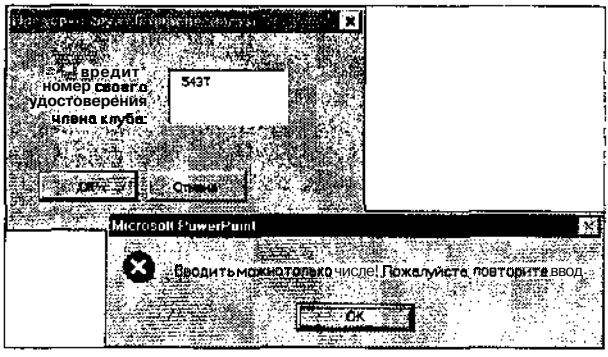


Рис. 10.27. Для проверки введенных данных в текстовом поле и отображения этого сообщения об ошибке использовалась процедура обработки события Change

Программный код проверки

Проверка значений элементов управления требует использования операторов `If...Then` и `Select Case`. В следующем простом примере проверяется на допустимость значение элемента управления кнопками прокрутки:

```
Private Sub spnVolumeControl_Change()
    If spnVolumeControl.Value = 13 Then
    MsgBox "13 не является допустимым значением."
    End If
End Sub
```

События для проверки правильности вводимых данных

Проверка правильности вводимых данных имеет смысл в любой момент взаимодействия пользователя с формой. В зависимости от выбранного момента проверка осуществляется в различных процедурах обработки различных событий, относящихся к элементу управления или к форме в целом, как показано в следующей таблице.

Момент проверки данных	Используемая процедура
При нажатии любой клавиши	Процедура обработки события `KeyPress` для элемента управления (для выяснения информации о нажатии отдельной клавиши)
При любом изменении значения элемента управления	Процедура обработки события `Change` (для оценки значения)
По окончании работы пользователя с элементом управления, непосредственно перед переходом к другому	Процедура обработки события `BeforeUpdate` (позволяет отменить обновление данных и вернуть пользователя к текущему элементу управления)
При закрытии формы пользователем	Процедура обработки события `Click` для кнопок OK или Закрыть в форме

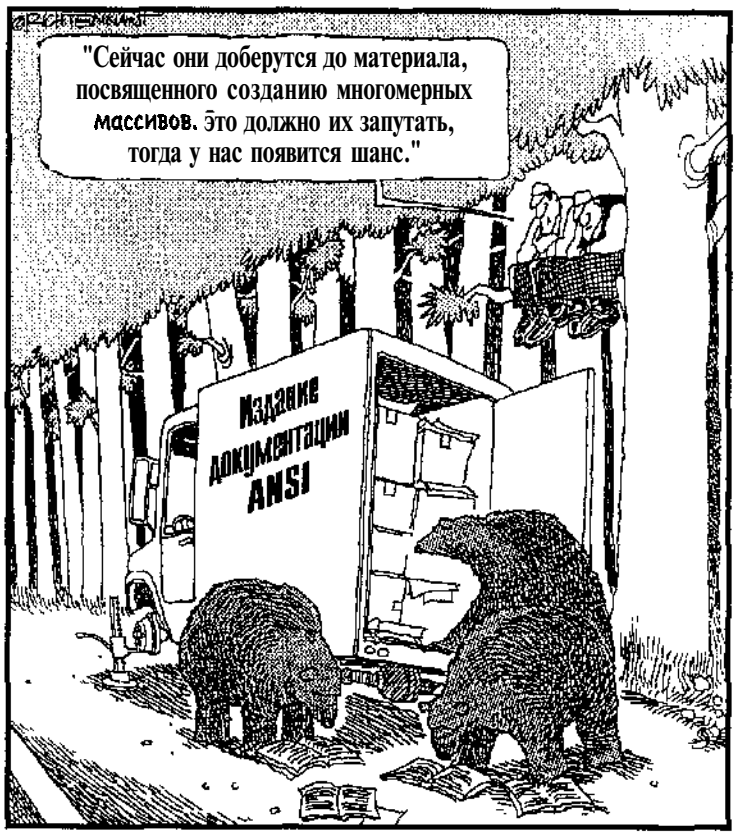
Как выбрать событие для проверки

Вы сами должны выбрать подходящую для проверки процедуру, ответив на следующие вопросы.

- ✓ Насколько быстро должен выполняться программный код проверки?
- ✓ В какой момент нужно информировать об ошибке пользователя — в момент появления неправильного значения или уже после того, как пользователь закончит формирование вводимых данных?
- ✓ Основывается ли проверка на значениях, полученных от нескольких элементов управления?

Часть III

Практикуемся в программировании на VBA



В этой части...

После того как вы познакомились с основами программирования на VBA и решили попробовать свои силы, вам пора изучить материал части III. В главе 11 рассматриваются встроенные функции и операторы VBA; я не рекомендую вам создавать собственную процедуру, если встроенная функция сможет выполнить те же действия всего в одну строку. В главе 12 мы поговорим об использовании объектов. В главе 13 мы еще раз затронем тему управления данными, однако на этот раз речь будет идти о таких сложных вопросах, как использование массивов и коллекций.

Глава 11

Инструменты встроенного оркестра VBA

В этой главе...

- Использование встроенных команд вместо нового изобретения программирования
- Безопасное форматирование данных с помощью функции `Format`
- Преобразование данных одних типов в другие
- Работа с текстовыми строками
- Забавы с датами и временем
- Использование математических и финансовых функций

Прежде чем создавать свою собственную процедуру с чистого листа, убедитесь, что вы не изобретаете велосипед. Каждая версия VBA приходит с небольшим арсеналом встроенных команд, предназначенных для решения самых разнообразных часто возникающих задач. В этой главе предлагается обзор наиболее полезных из таких готовых к бою единиц вооружения. Кое-что в этот обзор не вошло и обсуждается среди прочего в других главах.

Знакомство со встроенными командами

Похоже, VBA немного не соответствует тому имиджу, который он себе создает. Ох, как отчаянно VBA хочет выглядеть истинным *объектно-ориентированным* языком программирования (об объектно-ориентированном программировании мы поговорим в главе 12). С другой стороны, следует признать, что есть кое-что, что вы хотели бы иметь в программе, но нельзя сказать, что это естественным образом умещается в матрицу объектно-ориентированного подхода.

В своем решении, достойном всех наших депутатов, VBA помещает в метод объекта как минимум одну команду, которую было бы естественнее оставить независимой, и в то же время оставляет независимыми другие действия, которые вполне подходят для связывания их с объектами.

В обшем, учитывая, что некоторые действия могут оказаться в нелогичных для них категориях, вы получаете три типа встроенных VBA-команд, способных выполнять полезную работу.

- ✓ Операторы. Хотя термин *оператор* уже используется для обозначения целой программной директивы (см. главу 7), VBA называет операторами и отдельные ключевые слова, выполняющие специальные задачи. Некоторые из таких ключевых слов функционируют сами по себе как целые операторы. Например, оператор `Beep` заставляет динамик компьютера издавать звук. Другие же должны использоваться как составные части законченных операторов. Например, оператор `ChDir` (перейти в каталог) бесполезен без аргумента, задающего каталог или папку, куда нужно перейти:

```
ChDir("\Отчеты о мечтаниях")
```

- ✓ **Функции.** Встроенные функции работают точно так же, как и процедуры типа `Function`, обсуждавшиеся в главе 7,— в том смысле, что они возвращают значение. Использование функции часто заключается в присваивании ее значения переменной, как в следующем примере с функцией `Tan` (тангенс):

```
dblТангенс = Tan(dblЛюбойУгол)
```

Функции могут также обеспечивать значения для более сложных выражений или условных операторов, например:

```
If Tan(dblОстрыйУгол) < 45 Then
```

- ✓ **Методы встроенных объектов.** В этой группе курьезно выглядит метод `Print`, единственный для объекта Debug и предназначенный для направления вывода в окно немедленного выполнения команд (окно Immediate) в редакторе Visual Basic. Для этого используются операторы типа

```
Debug . Print (strПосланиеМарсианина)
```

- ✓ Лично я не вижу никаких преимуществ в связывании команды `Print` с объектом, но сам по себе метод `Print` оказывается очень полезным. Подробно он обсуждается в главе 14. Здесь же замечу, что, хотя VBA и не инкапсулирует файлы, как объекты, можно дополнительно установить программную надстройку, которая позволит управлять файлами и работать с их содержимым на основе объектно-ориентированного подхода (подробности — в главе 12).

В табл. 11.1 приводятся некоторые из встроенных VBA-команд, взятые для примера и относящиеся к различным категориям (операторы, функции, методы). Дальше в главе будут рассматриваться чаще всего используемые встроенные команды.

Таблица 11.1. Примеры встроенных функций, операторов и методов

Команда	Тип	Выполняемые действия
`Randomize`	Оператор	Инициализирует генератор случайных чисел
`Sqr(число)`	Функция	Возвращает значение квадратного корня числа
`Format(строка)`	Функция	Форматирует строку в соответствии с заданным описанием
`Date`	Оператор	Устанавливает системную дату
`Date`	Функция	Возвращает текущую системную дату
`Err . Raise`	Метод объекта `Err`	Генерирует ошибку выполнения с заданным кодом

Форматирование данных

VBA-функции `Format` форматируют данные любого из встроенных типов по указанному образцу для отображения их на экране или на печати. С помощью этих функций очень просто отображать значения дат в виде короткого (19.12.99), среднего (19-дек-99) или длинного формата даты (19 декабря 1999 г.), или любого из нескольких других предлагаемых VBA форматов (VBA хранит значения дат в виде совершенно неудобоваримых чисел). Подобные трюки можно выполнять и с числовыми значениями, и со строками. На самом деле, `Format` конвертирует предложенное вами значение в новую строку, добавляя в нее символы, необходимые для представления данных в нужном виде.

Функция Format предлагается и в VBA 5, и в VBA 6, однако VBA 6 может похвастать еще четырьмя новыми родственными функциями, каждая из которых предназначена для своего конкретного типа данных.

Работа с функцией Format

Функция Format, имеющаяся в VBA 5 и VBA 6,— невероятно гибкая. Она применима практически к любым типам данных и имеет настраиваемый вывод — если ни один из встроенных форматов не подойдет, вы можете создать свой, какой пожелаете.

В упрощенной форме синтаксис функции Format выглядит так (здесь опущены два необязательных и редко используемых аргумента, относящихся к датам, — о них вы можете узнать из справки VBA):

```
Format(выражение,"формат")
```

Аргумент выражение должен содержать выражение, представляющее число, строку или дату, которые вы хотите форматировать (т.е. отобразить в подходящей форме). Ясно, что этот аргумент обязателен.

Аргумент формат и является тем аргументом, который выполняет реальную работу. Как видно из следующего примера, аргумент формат нужно заключить в кавычки.

Чтобы использовать функцию Format, присвойте ее переменной либо свойствам Value или Caption элемента управления в форме. Например, оператор

```
lblDateMessage.Caption = "Сегодня " & _
    Format(Now, "Long date")
```

отображает текст "Сегодня 19 март 2001 г." в виде текста надписи элемента управления lblDateMessage (в предположении, что сегодня 19 марта 2001 года).

Использование встроенных форматов для функции Format

В табл. 11.2 приводятся именованные встроенные форматы для данных различных типов в VBA. Используйте их как аргумент в функции Format. Не забудьте при этом заключить в кавычки имя выбранного вами формата (из первого столбца таблицы).

Таблица 11.2. Именованные форматы для использования с функцией Format

Имя формата	Описание вывода	Пример (соответствует установке языка Русский в панели управления)
Числовой		
General Number	Число без выделения разрядов тысяч	2001,5599
Currency	Число с выделением разрядов тысяч и с двумя знаками после десятичного разделителя, а также с символом денежной единицы	2 001,56р.
Fixed	Как минимум один знак слева и два знака справа от десятичного разделителя (без выделения разрядов тысяч)	3390,10

Окончание табл. 11.2

Имя формата	Описание вывода	Пример (соответствует установке языка Русский в панели управления)
Standard	Число с выделением разрядов тысяч и как минимум одним знаком слева и двумя знаками справа от десятичного разделителя	1 323,45
Percent	Число, умноженное на 100, с двумя знаками справа от десятичного разделителя и знаком процентов (%) справа	12,54%
Scientific	Число в стандартном виде для научных расчетов	1,23E+02
Логический		
Yes/No	Нет, если значение равно 0, иначе Да	Да
True/False	Ложь, если значение равно 0, иначе Истина	Истина
On/Off	Выкл, если значение равно 0, иначе Вкл	Вкл
Дата/время		
General Date	Дата и/или время, в зависимости от значения, представленного в соответствии с установками в панели управления	15.03.98 17:27:45
Long Date	Дата в виде, заданном в панели управления для полного формата даты	15 Март 2001 г.
Medium Date	Дата в виде, заданном в панели управления для среднего формата даты	15-мар-01
Short Date	Дата в виде, заданном в панели управления для краткого формата даты	15.03.01
Long Time	Время (часы, минуты и секунды) в виде, заданном в панели управления для полного формата времени	17:27:45
Medium Time	Время (часы и минуты) в 12-часовом формате с метками до/после полудня, заданными в панели управления	05:27
Short Time	Время (часы и минуты) в 24-часовом формате	17:27

Создание собственных форматов

Можно создать свои собственные форматы, собрав их из символов, имеющих специальные значения для аргумента *формат*. Например, чтобы отобразить строки текста символами верхнего или нижнего регистров, используйте функцию Format с аргументами ">" или "<" соответственно. За неимением места я не могу описывать все эти специальные символы — вы найдете их самостоятельно, открыв сначала раздел Format Function (функция Format) в файле справки VBA, а затем щелкнув на ссылке See Also (см. также)

и просмотрев разделы User-defined Formats (пользовательские форматы). Но чтобы не совсем вас разочаровывать, я привожу следующий пример с парой пользовательских форматов в действии. Кроме того, этот же пример иллюстрирует использование еще одной встроенной функции VBA — Iif:

```
MsgBox "Сейчас " & Format(Now, "h:nn") &
    ". Это время " & IIf(Format(Now, "a/p") = "a",
    "до обеда.", "после обеда.")
```

Если вы выполните этот программный код до обеда, оператор отобразит на экране строку типа "Сейчас 9:07. Это время до обеда". После обеда вы получите нечто похожее на сообщение, показанное на рис. 11.1. (Кстати, функция MsgBox обсуждается ниже в этой же главе в разделе "Отображение окон сообщений".)

Рис. 11.1. Окно сообщения

Теперь объяснения. В первой функции Format аргументом формат является "h:nn". Символ h соответствует стандарту отображения часа одной цифрой, когда имеется в виду время до 10:00. После двоеточия nn задает отображение минут с незначащим нулем, если прошло меньше 10 минут после начала часа.

Вторая из используемых функций Format вложена в функцию IIf. Ввиду того, что аргументом формат в данном случае является "a/p", возвращаемыми значениями будут просто либо "a", либо "p", в зависимости от того, превышает время 12 часов дня или нет. Обычно a/p используется как компонент более длинной строки аргумента формат, но в данном случае результат функции видеть не требуется, и он обрабатывается функцией IIf.

Использование функции IIf

Функция IIf— это миниатюрная версия оператора If...Then (обсуждаемого в деталях в главе 12). Синтаксис этой функции следующий:

```
IIf (выражение, результатЕслиИстина, результатЕслиЛожь)
```

В переводе на "человеческий" язык функция работает следующим образом: если выражение есть истина, то функция возвращает значение результатЕслиИстина, если же выражение есть ложь, то возвращается значение результатЕслиЛожь.

В предыдущем примере выражением для проверки было

```
Format(Now, "a/p") = "a"
```

Поэтому если функция Format возвращает значение a, данное выражение есть Истина. Если же она возвращает любое другое значение, включая p, то выражение есть Ложь. Затем VBA вычисляет результат функции IIf на основе того, является ли значение выражения истиной или ложью. Если это значение Истина, функция IIfвозвращает до обеда, а если Ложь, то после обеда.

Работа с другими функциями форматирования

В VBA 6 были добавлены несколько функций, подобных функции Format, — FormatNumber, FormatDateTime, FormatCurrency и FormatPercent — для работы сданными соответствующих типов. Подобно оригинальной функции Format, каждая из этих функций-наследниц возвращает строку с форматированным представлением исходного значения. Ввиду того, что эти новые функции по умолчанию возвращают результаты, нужные вам чаще всего, эти функции могут оказаться в использовании удобнее, чем сама функция Format. Вкратце их работу можно описать следующим образом.

- ✓ FormatNumber(число,ЧислоЗнаковПослеЗапятой, ОтображатьНезначащийНуль, ИспользоватьСкобкиДляОтрицательныхЧисел, ГруппироватьРазряды). Возвращает значение **числа** — единственного обязательного **аргумента** — в виде форматированной строки. Если опустить необязательные аргументы, функция FormatNumber отформатирует число на основе установок по умолчанию, заданных на вкладке Числа окна контрольной панели Язык и стандарты Windows. В предположении, что языком Windows выбран русский, вы получите число вида 132 328,55. Необязательные аргументы функции FormatNumber позволяют изменить заданные по умолчанию значения для числа знаков после запятой, отображения незначащего нуля перед запятой для чисел, меньших 1, представления отрицательных чисел в скобках и отделения групп разрядов одной от другой (например, пробелами).
- ✓ FormatDateTime (дата, формат). Конвертирует значение даты в форматированную строку. Если аргумент формат не указан, строка форматируется в соответствии с системными установками для краткого формата даты и длинного формата времени, заданными на вкладках Дата и Время панели управления Язык и стандарты Windows. Здесь аргумент формат является числом (а не строкой, как в функции Format), но его можно задавать, используя именованные константы вида vbLongDate или vbShortTime.
- ✓ FormatCurrency (число). Возвращает значение числа, отформатированного как денежное, в соответствии с текущими установками в панели управления. В остальном функция FormatCurrency работает подобно функции FormatNumber и имеет те же необязательные аргументы.
- ✓ FormatPercent (число). Умножает число на 100 и добавляет знак процентов (например, 0,05 превращается в 5,00%). Функция FormatPercent имеет те же необязательные аргументы, что и функция FormatNumber. Чтобы не выводить знаки дробной части, задайте аргумент ЧислоЗнаковПослеЗапятой равным 0 (например, FormatPercent (.05, 0)).

Конвертирование данных

Как упоминалось в главе 9, VBA автоматически конвертирует данные одних типов в другие "на лету". Такие автоматические преобразования, конечно, удобны, да и получаемые при этом результаты обычно соответствуют желаемым. Правда, при этом нужно обязательно подчеркнуть слово *обычно*.

Когда автоматическое преобразование вас не устраивает, VBA готов предложить массу функций для явного преобразования данных одних типов в другие. Эти функции можно использовать для того, чтобы

- ✓ быть уверенным, что VBA выполняет именно то преобразование, которое нужно;
- ✓ выполнить преобразования, которые не выполняются в VBA автоматически;
- ✓ сделать свой программный код яснее.

Для конвертирования данных в VBA предусмотрена целая группа функций — `CBool`, `CByte`, `CCur` и т.д. — своих для каждого из встроенных типов данных, за исключением `Object`. Например, в результате выполнения оператора `boolMaybe = CBool(123)` переменная `boolMaybe` будет содержать значение `True` (вообще любое число, не равное нулю, в результате даст `True`, наверное потому, что любое такое число ассоциируется с чем-то существующим).

Замечу, однако, что, кроме превращения самой операции конвертирования в явную, эти функции не делают ничего такого, что не может предложить VBA при автоматическом конвертировании. (Об особенностях функции `CDec` см. в главе 9.)

Функции `Fix` и `Int` отбрасывают дробную часть любого заданного им числа, возвращая целое число. Но, в отличие от функций `CInt` и `CLng`, они не выполняют привычного округления — например, `Int(4.989)` возвращает 4, а не 5.

Результаты этих двух функций отличаются только при обработке отрицательных чисел. Тогда `Int` возвращает ближайшее к значению аргумента меньшее целое число, а `Fix` просто отбрасывает дробную часть аргумента.

Работа с шестнадцатеричными и восьмеричными значениями

Функции `Hex` и `Oct` преобразуют стандартные десятичные целые числа в строки, содержащие соответственно их шестнадцатеричные и восьмеричные эквиваленты. Возможность явного конвертирования в обратном направлении в VBA не предусмотрена, но зато вы *можете* напечатать буквальные шестнадцатеричные или восьмеричные значения и позволить редактору Visual Basic превратить их в десятичные автоматически. При этом любой шестнадцатеричный или восьмеричный литерал должен предваряться специальным кодом: шестнадцатеричный — кодом `&H`, а восьмеричный — кодом `&O` (это буква O, а не цифра 0). Например, в результате выполнения оператора `intBases = 10+&O12 + &HA` значением переменной `intBases` будет число 30, как и должно быть (это очевидно).

Преобразование чисел в строки и наоборот

Некоторые функции VBA конвертируют числа в строки. К таким функциям относятся следующие.

- ✓ `CStr`. Превращает данные любых типов (кроме типа `Object`), включая числовые, в соответствующую им строку. Вывод форматируется в соответствии с установками, заданными в панели управления Язык **и стандарты** Windows. Например, во Франции `CStr(200.02)` на выходе выдаст строку `"200,02"`. Здесь следует заметить, что хотя результирующая *строка* и соответствует региональным установкам, подаваемые на вход числовые данные должны быть представлены в формате, соответствующем английскому языку США.
- ✓ `Str`. Конвертирует число в строку, но всегда форматирует строку в соответствии со стандартом английского языка США с точкой в качестве десятичного разделителя.
- ✓ **Функции конвертирования в числовые типы данных.** Конвертируют строки в соответствующие числовые значения, но только тогда, когда все символы в строке рас-

познаются как допустимые для чисел. В данном случае тоже все зависит от установок в панели управления. Например, в России CDbl("200,02р.") в результате дает 200,02; в США тот же оператор порождает ошибку, но CDbl("$200.02") прекрасно работает.

- ✓ Val. Конвертирует числа в строках в числовые значения, останавливаясь там, где встречается первый символ, недопустимый для чисел. Независимо от установок панели управления, распознает только цифры и десятичную точку (а не запятую, например). Однако игнорирует пробелы, символы табуляции и переходы на новую строку. Так, выражение Val("28 190.43 12 by 14") в результате даст 28190,4312.
- ✓ Chr. Конвертирует числовой ANSI-код в соответствующий символ. Используйте эту функцию, когда в строку нужно поместить символ, который нельзя напечатать.
- ✓ **Asc.** Примерно соответствует обратной к функции Chr — возвращает числовой код первого символа в строке.

Работа со строками

В VBA имеется довольно богатая коллекция операторов и функций для форматирования строк и извлечения тех их частей, которые вы сочтете особенно привлекательными. В табл. 11.3 я представил все относящиеся к строкам команды, которые мне удалось обнаружить.

Таблица 11.3. Операторы и функции для обработки строк

Не забывайте, что буквальные строковые значения (в отличие от переменных, содержащих строковые значения) должны заключаться в кавычки.

Оператор или функция	Тип	Выполняемые действия (для операторов) или возвращаемые значения (для функций)
Asc (строка)	Функция	Числовой код первого символа в строке
Chr (код_символа)	Функция	Символ, соответствующий значению параметра код_символа
Filter (массив_источник, эталон, включает, метод_сравнения)	Функция (только в VBA 6)	Массив, состоящий только из тех строк из массива-источника, которые содержат эталон. Аргумент массив_источник должен быть массивом строковых значений. Если необязательный аргумент включает равен False, то функция возвратит только строки, которые не включают эталон
Format (строка)	Функция	См. выше раздел "Форматирование данных"
Hex (число)	Функция	Строка, содержащая шестнадцатеричное представление числа
InStr (старт, строка1, строка2)	Функция	Число, соответствующее позиции строки 2 в строке 1; поиск начинается с позиции старт, этот аргумент не обязателен
InStrRev (строка1, строка2, старт)	функция (только в VBA 6)	Число, соответствующее позиции строки 2 в строке 1, считая с правого края строки 1; поиск начинается с позиции старт, этот аргумент не обязателен

Оператор или функция	Тип	Выполняемые действия (для операторов) или возвращаемые значения (для функций)
Join(массив_строк, разделитель)	функция (только в VBA 6)	Одна строка, комбинирующая все строки из массив_строк, который должен быть массивом строковых данных. По умолчанию Join вставляет пробел между строками, взятыми из массива строк, но можно задать и другой символ разделителя (символ необходимо заключить в кавычки)
Left (строка, длина)	Функция	Строка указанной длины из символов, взятых подряд из строки, задаваемой аргументом строка, начиная с левого края последней
Len(строка)	Функция	Число символов в строке
LCase (строка)	Функция	Копия строки, представленная символами нижнего регистра
LSet строковая_переменная = строка	Оператор	Устанавливает значение строковой переменной, равное заданной строке, без изменения длины этой строковой переменной и с размещением заданной строки начиная с левого края переменной
LTrim(строка)	Функция	Новая строка, содержащая копию данной строки без пробелов в начале
Mid(строка, старт, длина)	Функция	Новая строка заданной длины из символов данной строки, взятых подряд, начиная с позиции старт
Mid(строковая_переменная, старт, длина) = строка	Оператор	Начиная с позиции старт заменяет символами данной строки заданное аргументом длина число символов в строковой переменной
Oct(число)	Функция	Строка, содержащая восьмеричное представление числа
Replace(строка, найти, заменить, старт, число_замен, метод_сравнения)	Функция (только в VBA 6)	Новая строка, получаемая в результате замены в заданной строке текста найти текстом заменить. Аргументы старт, число_замен и метод_сравнения не обязательны. Используйте число_замен, чтобы указать, сколько замен следует сделать, если текст найти встречается несколько раз (по умолчанию заменяются все случаи появления текста найти в строке}
Right (строка, длина)	Функция	Строка указанной длины из символов, взятых подряд из строки, задаваемой аргументом строка, начиная с правого края последней
RSet зтроковая_переменная = строка	Оператор	Устанавливает значение строковой переменной, равное заданной строке, без изменения длины этой строковой переменной и с размещением заданной строки с правого края переменной
RTrim(строка)	Функция	Новая строка, содержащая копию данной строки без пробелов в конце
Space (число)	Функция	Строка, состоящая из указанного числа пробелов

Окончание табл. 11.3

Оператор или функция	Тип	Выполняемые действия (для операторов) или возвращаемые значения (для функций)
`Split(строка, разделитель, предел, метод_сравнения)`	Функция (только в VBA 6)	Массив строк, полученный в результате разделения заданной строки. По умолчанию оригинальная `строка` разделяется по пробелам, но можно задать `разделитель`, отличный от пробела (для задания разделителя используйте символ, заключенный в кавычки). Необязательный аргумент `предел` задает максимальное число строк в возвращаемом массиве
`StrComp(строка1, строка2)`	Функция	0, если две строки равны; -1, если `строка1` меньше, чем `строка2`; 1, если `строка1` больше, чем `строка2` (о сравнении строк говорилось в главе 9)
`StrConv(строка, метод_перевода)`	Функция	Новая строка, созданная на основе заданной указанным методом перевода
`String(число, символ)`	Функция	Строка, состоящая из заданного числа повторяющихся символов
`StrReverse (строка)`	Функция (только в VBA 6)	Строка, содержащая символы заданной строки в обратном порядке
`Trim(строка)`	Функция	Новая строка, содержащая копию данной строки без пробелов в начале и конце
`UCase(строка)`	Функция	Копия строки, представленная символами верхнего регистра

Не расстраивайтесь, если не обнаружите строковую функцию, которая делает в точности то, что вам нужно. Воспринимайте эти функции как строительные блоки — часто для достижения нужного результата эти функции приходится вкладывать одну в другую.

Отбрасывание символов в конце строк с помощью Len и Left

Представим, например, что у вас есть список полных имен в виде следующих строк:

```
"Самодур, Иван Акакиевич, акад."
"Невеселый, Степан Федорович, проф."
"Шутник, Василий Степанович, канд."
```

(такой список должен храниться в *массиве* строк; массивы будут рассматриваться в главе 13).

Предположим, что кто-то заставил вас убрать звания в конце каждой из вышеприведенных строк. Имена в списке имеют разную длину, так что готовые функции задачу не решают. Но если предположить, что оригинальное имя содержится в переменной `strСтарая`, то поможет следующий трюк:

```
strНовая = Left(strСтарая, Len(strСтарая) - 7)
```

Если бы VBA читал книги вместо программного кода, я объяснил бы ему, что эти действия можно выполнить следующим образом.

1. **Сначала вызывается функция Len, которая вычисляет длину оригинальной строки.**
2. **Из результата, полученного в п. 1, вычитается 7, поскольку каждое из званий занимает семь символов в конце строки (включая запятую и пробел).**
   В результате получится число, которому должна быть равна длина новой строки.
3. **Вызывается функция `Left`, использующая вычисленную длину в качестве значения второго аргумента.**
   Функция `Left` читает столько символов, сколько задано вторым аргументом, из строки, указанной первым аргументом, возвращая только эти прочитанные символы.
4. **Новая, более короткая строка, возвращенная функцией `Left`, присваивается переменной `strНовая`.**

Извлечение символов из строк

Продолжая работать с новым списком уже без званий, предположим, что теперь вас попросили извлечь имена с отчествами и поместить их в новый список. Эту задачу можно выполнить с помощью следующего оператора:

```
strИмя = Mid(strНовая, InStr(strНовая, ",") + 2)
```

Функция Mid извлекает символы из любого места в строке, которую вы укажете в виде аргумента. Второй же аргумент этой функции говорит, с какого места в этой строке следует начать извлечение символов. В данном случае этот аргумент представляет собой выражение, состоящее из функции InStr плюс 2.

Функция InStr ищет в заданной строке (первый аргумент) другую строку (второй аргумент). В данном случае второй аргумент — это запятая, которая в списке следует за фамилией. Значение, которое возвращает InStr, соответствует номеру позиции в первой строке, где встречается вторая строка, — теперь вы знаете, где заканчивается фамилия.

К значению, возвращенному функцией InStr, добавляется 2, чтобы пропустить запятую и следующий за ней пробел и найти позицию, с которой начинается имя. Теперь функция Mid извлечет символы, начиная с этого места и до конца строки (поскольку не указан третий, необязательный аргумент `длина`).

Работа с датами и временем

В реальном программировании работе с датами и временем отводится далеко не последняя, а часто и главная роль. Поэтому VBA предлагает ряд операторов и функций, позволяющих выяснить текущие время и дату, проводить вычисления с датами и извлекать из переменных со значениями дат различные компоненты типа времени, года или дня недели.

В табл. 11.4 представлено описание таких команд. У некоторых функций в этой таблице опущены необязательные аргументы — вы всегда сможете уточнить детали по справке VBA. После таблицы несколько наиболее важных команд, относящихся к датам и времени, рассматриваются подробнее.

Чтобы воспользоваться предоставляемым здесь материалом, вы должны ясно представлять, как VBA обрабатывает значения дат и времени и как работать с переменными, хранящими даты (см. главу 7).

Таблица 11.4. Операторы и функции для работы с датами и временем

При задании аргументов не забывайте заключать буквальные значения дат в пару символов #, а строковые значения типа интервалов дат — в кавычки.

Имя	Тип	Выполняемые действия (для операторов) или возвращаемые значения (для функций)
Date	Функция	Текущая системная дата
Date (дата)	Оператор	Устанавливает системную дату по значению аргумента дата
DateAdd (интервал, число, дата)	Функция	Новое значение даты, равное сумме исходной даты и казенного числа заданных интервалов даты или времени
DateDiff (интервал, дата1, дата2)	Функция	Число указанных интервалов даты или времени, помещающихся в отрезке времени между датами 1 и 2
DatePart (интервал, дата)	Функция	Целое значение, представляющее указанный интервал даты
DateSerial (год, месяц, день)	Функция	Значение даты, определяемое аргументами год, месяц, день, которые должны быть числовыми
DateValue (строка_с_датой)	Функция	Значение даты, соответствующее строке с датой
Day (дата)	Функция	Целое значение, соответствующее дню месяца, заданного датой
Hour (время)	Функция	Целое значение между 0 и 23 включительно, представляющее час суток, заданный указанным временем
Minute (время)	Функция	Целое значение между 0 и 59 включительно, представляющее минуты, заданные указанным временем
Month (дата)	Функция	Целое значение между 0 и 12 включительно, представляющее месяц, заданный указанной датой
MonthName (месяц, сократить)	Функция (только в VBA 6)	Строка, содержащая название месяца, соответствующего аргументу месяц, который должен быть числом от 1 до 12. Если необязательный аргумент сократить указан и равен True, то возвращаемая строка будет содержать сокращенное название месяца
Now	Функция	Значение, представляющее текущие системные дату и время
Second (время)	Функция	Целое значение между 0 и 59 включительно, представляющее секунды, заданные указанным временем
Time	Функция	Значение, представляющее текущее системное время
Time (время)	Оператор	Устанавливает системное время по значению аргумента время
Timer	Функция	Число секунд, прошедших с полуночи

Окончание табл. 11.4

Имя	Тип	Выполняемые действия (для операторов) или возвращаемые значения (для функций)
`TimeSerial (часы, минуты, секунды)`	Функция	Значение времени, заданное аргументами `часы`, `минуты`, `секунды`
`TimeValue (строка_с_временем)`	Функция	Значение времени, соответствующее строке со временем (вся информация о дате будет отброшена)
`Weekday (дата)`	Функция	Целое число, представляющее день недели, соответствующий указанной дате
`WeekdayName (день_недели, сократить, первый_день_недели)`	Функция (только в VBA 6)	Строка, содержащая название дня недели, заданного аргументом `день_недели`, который должен быть целым числом от 1 до 7. Если необязательный аргумент `сократить` указан и равен `True`, то возвращаемая строка будет содержать сокращенное название дня недели. Нумерацию дней недели можно менять, задавая необязательный аргумент `первый_день_недели` равным константам типа vbMonday, vbTuesday и т.д.
`Year(дата)`	Функция	Целое число, представляющее год, соответствующий указанной дате

Дата со временем

В VBA есть несколько простых операторов и функций для работы с *системной датой* и *системным временем,* т.е. со значением даты и времени, которое изменяется и хранится встроенными часами компьютера.

Функция Now возвращает текущие системные дату и время в формате переменной типа Data, как, например, в случае `datЛовиМомент = Now`.

Чтобы обеспечить работу с системным временем и системной датой по отдельности, VBA предлагает по паре отдельных команд для каждого из этих двух элементов системы. Несколько неожиданным при этом оказывается то, что в каждой из пар и оператор, и функция имеют одинаковые имена.

Например, чтобы *извлечь* (т.е. получить) системное время в формате даты VBA, нужно использовать *функцию* Time, а чтобы *установить* (т.е. задать) системное время, необходимо использовать *оператор* Time. Другими словами, ключевое слово Time выполняет различные действия в зависимости от контекста:

```
datЭтоМоеВремя = Time ' функция Time возвращает значение
' системного времени
Time (#3:15 AM#) ' оператор Time устанавливает системное время
```

Подобным образом работают и различаются соответствующие функция Date и оператор Date.

Исчисление дат

Как вы знаете, время — понятие относительное. Поэтому главное — получить дату и время в таком виде, чтобы сравнить их с другими датами и временем. Иногда нужно знать, как далеко

отстоят одна дата от другой или одно время от другого. А иногда бывает нужно знать, какая дата будет, скажем, через два года и три месяца. Несколько предлагаемых в VBA функций для обработки значений дат и времени превращают вычисления с датами в совсем простое дело.

Сложение и вычитание значений дат и времени

Используйте функцию DateAdd, когда нужно узнать, какая дата наступит через три года или что было на часах один час и пятнадцать минут тому назад. Такие вычисления можно выполнить и без компьютера вообще, но они требуют особой внимательности, поскольку операции с компонентами значений дат и времени не подчиняются правилам привычной арифметики десятичных чисел (дней недели — 7, секунд — 60, и т.д.), да и самих компонентов наберется немало.

Функция DateAdd имеет следующий синтаксис: DateAdd (интервал, число, начальная_дата).

Возвращаемое функцией значение равно значению, полученному в результате добавления к начальной дате (или начальному времени) заданного числа интервалов. Если нужна разность, а не сумма, задайте для аргумента число отрицательное значение. В табл. 11.5 приведен набор символьных кодов, которые допустимы для аргумента интервал (при использовании не забудьте заключить соответствующий код в кавычки).

Таблица 11.5. Значения аргументов функций DateAdd **и** DatDiff

Значение аргумента (заключается в кавычки)	Задаваемый интервал
yyyy	Год
q	Квартал
га	Месяц
y	День года
d	День
w	День недели
ww	Неделя
h	Час
п	Минута
s	Секунда

Например, в результате выполнения оператора

```
datБыло = DateAdd ("s", -90, Now)
```

переменная datБыло будет содержать значение, соответствующее времени, которое было 90 секунд тому назад относительно текущего времени. Если вам нужно отобразить на экране только ту часть этого значения, которая относится ко времени, используйте функцию Format с соответствующим времени аргументом форматирования (например, Format (datБыло, "Medium time")), как было описано в разделе "Форматирование данных".

При этом функция DateAdd не возвращает недопустимых значений дат: если добавить один месяц к дате 31 августа, то результатом будет 30 сентября. Точно так же функция DateAdd не забудет, если нужно, изменить в результате и год.

С помощью функции DateAdd можно добавлять или вычитать некоторое число однотипных интервалов, но, например, для выяснения даты, которая была два года и три месяца назад, ничто не мешает применить эту функцию дважды.

В качестве альтернативы VBA предлагает для дат и времени две другие функции, которые позволяют смешивать интервалы,— DateSerial и TimeSerial. Каждая из этих функций имеет по три аргумента;

```
DateSerial (год, месяц, день)
TimeSerial (часы, минуты, секунды)
```

Все аргументы в данном случае обязательны и должны быть выражениями, дающими *целые* значения (а не значения типа Date). Посмотрите, можете ли вы сказать, что будет получено в результате выполнения следующего фрагмента программного кода, и обратите внимание на вычисления в третьем аргументе:

```
intГод = 1999
intМесяц = 12
intДень = 3 1
datНовоеТысячелетие = DateSerial (intГод, intМесяц, intfleHb + 1)
```

И, конечно же, любые из этих аргументов или даже все могут быть просто числами.

Вычисление разности двух дат

Используйте функцию DateDiff, чтобы выяснить, сколько определенных интервалов времени (годы, месяцы, недели и т.п.) помещается между двумя заданными датами или моментами времени. Например:

```
lngСколько = DateDiff ("m", #2/12/90#, #10/12/01#)
```

В результате выполнения этого оператора переменная lпдСколько — переменная типа Long — будет содержать число месяцев, прошедших между двумя указанными датами. Для первого аргумента функция DateDiffтребует код интервала времени, и в данном случае "m" сообщает функции о том, что нужно измерять в месяцах, а не в годах, неделях или каких-либо других единицах.

Вычислениевозрастаперсоны

Представленная ниже процедура типа Function использует функции DateDiff и DateSerial для вычисления возраста персоны в годах. Чтобы избежать появления ошибок при выполнении программы, можете добавить в программный код проверку типа данных того значения, которое получит аргумент, задающий дату рождения (используйте для этого функцию IsDate), а также проверку того, что полученная дата рождения не находится в будущем.


```
Function WhatAge(dateDateOfBirth As Date)

Dim intAge As Integer ' задайте Long, чтобы допустить
возраст > 255

intAge = DateDiff("yyyy", dateDateOfBirth, Date)

' проверяется, день рождения в этом году был или нет:
If DateSerial(Year(Date), Month(dateDateOfBirth), _
     Day(dateDateOfBirth)) > Date Then
          intAge = intAge - 1
End If

WhatAge = intAge

End Function
```

Простые контакты с внешним миром

Из-за, так сказать, паразитического образа жизни на больших приложениях, VBA-программы зачастую не требуют никаких элементов пользовательского интерфейса (диалоговых окон, меню и т.п.). При запуске VBA-программа может выполняться на базе активного документа приложения, сообразуя свои действия с содержимым этого документа.

Ясно, правда, и то, что существует немало ситуаций, когда требуется прямое взаимодействие с конечным пользователем VBA-программы. Прекрасные возможности для создания интерактивных окон в программе предлагают средства создания пользовательских форм VBA (о чем мы говорили в главе 10), но если есть возможность обойтись более простыми средствами, то чем проще, тем лучше.

Две VBA-функции, `MsgBox` и `InputBox`, обеспечивают неплохие возможности базового уровня для передачи пользователю сообщений и получения от него ответов.

- ✓ Функция MsgBox отображает окно с сообщением, но не только — она позволяет определить, на какой из двух (или больше) кнопок щелкнул пользователь, чтобы закрыть окно.
- ✓ Функция `InputBox` отображает окно с сообщением и полем, в котором пользователь имеет возможность напечатать ответ.

Отображение окон сообщений

Формально синтаксическая конструкция функции MsgBox выглядит так:

```
MsgBox(сообщение[, кнопки] [, заголовок] [, файл_справки, раздел])
```

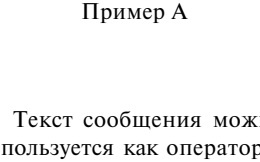
Как показывают скобки, обязательный здесь только аргумент `сообщение`, который, очевидно, задает то сообщение, которое вы желаете отобразить на экране.

Задание текста сообщения

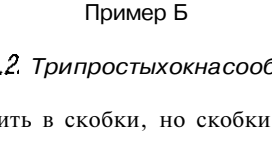
В своей простейшей форме функция MsgBox действует как оператор. От вас требуется только напечатать его в отдельной строке и добавить сообщение, которое требуется отобразить. Например:

```
MsgBox "Это проверка работы MsgBox."
```

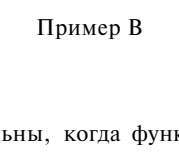
В результате выполнения такой строки VBA отобразит на экране окно с сообщением, подобное показанному на рис. 11.2 (пример А).



Пример А



Пример Б



Пример В

Рис. 11.2. Три простых окна сообщения

Текст сообщения можно заключить в скобки, но скобки необязательны, когда функция используется как оператор.

Сообщение может быть также переменной или выражением. Как обычно, VBA автоматически конвертирует за вас числовые значения и даты в отображаемые символы. Например, следующий фрагмент программного кода вполне работоспособен, а результат показан на рис. 11.2 (пример Б):

```
intWishCount = 3
datWhen = Format(Now, "Short date")
strInfo1 = "На "
strInfo2 = " осталось желаний: "
MsgBox strInfo1 & datWhen & strInfo2 & intWishCount
```

Чтобы отобразить сообщение в несколько строк, добавьте в сообщение символы перехода на новую строку (их ASCII-код равен 13) с помощью функции Chr (пример В на рис. 11.2).

```
MsgBox "Это первая строка." & Chr(13) & "Это вторая строка."
```

Можно разбить текст многострочного сообщения и на столбцы, используя символы табуляции (их ASCII-код равен 9).

Создание непростых окон сообщений

Кроме сообщения, в окне сообщения может присутствовать одна из нескольких стандартных пиктограмм, а также несколько кнопок стандартного вида. Все это задается указанием одного числового значения для необязательного аргумента кнопки.

С пиктограммой ваше окно сообщения будет выглядеть немного "круче", чем простенькие окна с рис. 11.2. Показанное на рис. 11.3 окно сообщения содержит пиктограмму критической ситуации (эта пиктограмма обычно вызывает у пользователя некоторое волнение).

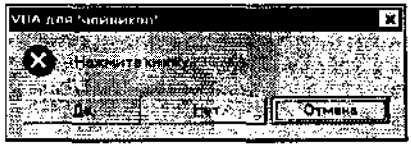


Рис. 11.3. Непростое окно сообщения

По умолчанию окно сообщения имеет только кнопку OK, но можно в него добавить и кнопки с надписями Отмена, Да, Нет, Стоп, Повтор и Пропустить в различных комбинациях. На рис. 11.3 как раз показано окно с одной из таких комбинаций кнопок.

Вычисление значения для аргумента кнопок

Как и для подобных аргументов многих других функций, значение для аргумента кнопки вычисляется как сумма констант, представляющих наборы кнопок и пиктограммы. Можно подсчитать нужное значение вручную, но лучше создать подходящее выражение из именованных VBA-констант, предлагаемых как раз для этих целей. В табл. 11.6 эти константы приведены вместе с их числовыми значениями и описаниями назначения.

Таблица 11.6. Константы VBA для окон сообщений и окон ввода

Константа	Числовое значение	Соответствующее действие
vbOKCnly	0	Отображение только кнопки OK
vbOKCancel	1	Отображение кнопок **OK и Отмена**
vbAbcrtRetryIgnore	2	Отображение кнопок Стоп, **Повтор и Пропустить**

Окончание табл. 11.6

Константа	Числовое значение	Соответствующее действие
vbYesNoCancel	3	Отображение кнопок Да, Нет и Отмена
vbYesNo	4	Отображение кнопок Да и Нет
vbRetryCancel	5	Отображение кнопок Повтор и Отмена
vbCritical	16	Отображение пиктограммы Критическое сообщение
vbQuestion	32	Отображение пиктограммы Предупреждающий запрос
vbExclamation	48	Отображение пиктограммы Предупреждающее сообщение
vbInformation	64	Отображение пиктограммы Информирующее сообщение
vbDefaultButton1	0	Выбор по умолчанию первой кнопки
vbDefaultButton2	256	Выбор по умолчанию второй кнопки
vbDefaultButton3	512	Выбор по умолчанию третьей кнопки
vbDefaultButton4	768	Выбор по умолчанию четвертой кнопки

С помощью этой таблицы несложно подсчитать, что при вызове функции, отобразившей окно сообщения, показанное на рис. 11.3, значение аргумента кнопки было равным 531. Но кому хочется это значение считать? Легче напечатать следующий оператор:


```
intA = MsgBox("Нажмите кнопку", vbYesNoCancel + vbCritical
+ vbDefaultButton3, "VBA для 'чайников'")
```

Третья из констант, использованных в выражении для аргумента кнопки, константа vbDefaultButton3, назначает третью кнопку (считая слева направо) кнопкой, выбираемой по умолчанию. В данном случае это кнопка Отмена. Если внимательно рассмотреть рис. 11.3, можно увидеть, что кнопка Отмена получила фокус ввода — надпись на ней очерчена пунктирной рамкой, и это означает, что именно эта кнопка будет активизирована, если вы нажмете клавишу пробела или <Enter>.

Какая кнопка нажата?

Смысл отображения кнопок в окне сообщения заключается в предоставлении пользователю возможности выбора действия. В данном случае пользователю не нужно ничего печатать — следует только щелкнуть на одной из кнопок. А вам нужно суметь выяснить, *на какой* из кнопок щелкнул пользователь.

Это просто, поскольку функция MsgBox возвращает целое значение, как раз соответствующее той кнопке, на которой щелкнул пользователь. Чтобы не нагружать свою память, вы можете сравнивать возвращаемое значение не с конкретными числами, а с предлагаемыми VBA именованными константами. Вот эти константы вместе с их *действительными* значениями.

Константа	Значение
vbOK	1
vbCancel	2
vbAbort	3
vbRetry	4
vbIgnore	5
vbYes	6
vbNo	7

Если в окне сообщения всего две кнопки, для выяснения, на какой из кнопок был щелчок, прекрасно подходит оператор If ... Then. Например:

```
If Msg3ox ("Начинать?", vbYesNo) = vbYes Then
    ЧтоНибудьСделать
Else
    НеДелать Ничего
End If
```

Когда кнопок больше, приходится использовать оператор If...Then...Else If.

Добавление заголовка

По умолчанию окно сообщения отображает в строке заголовка имя того VBA-приложения, из которого выполняется программа (см. рис. 11.2). Этот заголовок можно изменить на любой другой, задав соответствующее значение аргумента заголовок в операторе вызова функции MsgBox (см. рис. 11.3).

Получение информации от пользователя

Если вам нужна от пользователя более конкретная информация, не укладывающаяся в рамки выбора из трех кнопок, используйте функцию InputBox. Вот ее формальная синтаксическая конструкция, из которой удалены некоторые необязательные аргументы, расширяющие возможности:

```
InputBox(сообщение[, заголовок]  [, ответ])
```

Как видно из рис. 11.4, эта функция отображает диалоговое окно с полем текста, куда пользователь может впечатать некоторую, как предполагается, важную информацию. Чтобы передать эту информацию программе, присвойте возвращенное функцией InputBox значение строковой переменной:

```
strB = InputBox("Какие места предпочитаете?", "TWA", "У прохода")
```

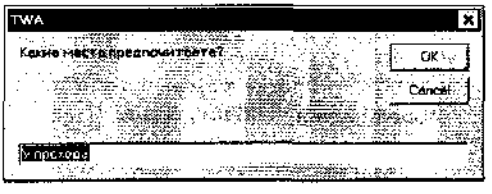


Рис. 11.4. Простое окно ввода

Хотя с помощью окна ввода можно получить больше информации, чем с помощью окна сообщения, с окном ввода программисту работать легче — тут нет кнопок и пиктограмм, которые можно выбирать. Аргументы `сообщение` и `заголовок` точно такие же, как и в случае функции `MsgBox`. Вы можете облегчить задачу пользователю, задав значение аргумента `ответ`. Тем самым вы предложите ему ответ, принимаемый по умолчанию, — если предложенный вами ответ подойдет, пользователь сможет просто нажать клавишу <Enter>.

Работа с логическими значениями

В случае переменной типа `Boolean` изменить ее значение на противоположное проще всего с помощью операции логического отрицания `Not`. При этом результат применения операции нужно просто присвоить той же самой переменной. Например, оператор

```
boolBlinking = Not(boolBlinking)
```

делает значение переменной `boolBlinking` равным `True`, если значением было `False`, и, наоборот, равным `False`, если значением было `True`.

Забавы с математикой и деньгами

В VBA предлагается целый ряд функций для работы с числовыми значениями. Эти функции предназначены для решения ряда задач от совсем простых (например, возвращение абсолютного значения или знака числа) до достаточно сложных вычислений алгебраических и тригонометрических величин. А добрая дюжина финансовых функций VBA поможет удержать на плаву любой утопающий бизнес.

Я не собираюсь останавливаться здесь на всех этих функциях, а рассмотрю лишь, как исключение, несколько избранных из них. Да и эти последние я обсуждаю только потому, что их непросто обнаружить в системе справки VBA и поэтому вы могли бы иначе просто не узнать о них. Внимательно ознакомьтесь с приведенным ниже списком, чтобы знать, что в VBA есть, а чего — нет.

Математические функции

В табл. 11.7 приведены функции VBA, которые можно назвать математическими. Перед тем как использовать их, не забудьте снова просмотреть раздел "Конвертирование данных" выше в этой же главе.

Таблица 11.7. Математические функции VBA

Функция	Возвращаемое значение
`Abs(число)`	Абсолютное значение числа
`Atn(число)`	Арктангенс числа
`Cos(число)`	Косинус числа
`Exp(число)`	Число e в степени, равной заданному числу
`Fix(число)`	Целая часть числа (см. описание функции `Int`)

Окончание табл. 11.7

Функция	Возвращаемое значение
`Int(число)`	Целая часть числа. Функции `Int` и `Fix` по-разному действуют только на отрицательные числа: `Int` возвращает ближайшее меньшее целое, а `Fix` просто отбрасывает дробную часть числа
`Log(число)`	Натуральный логарифм числа, значение двойной точности
`Rnd(число)`	Случайное число, значение одинарной точности
`Sgn(число)`	1, если `число` положительно, 0, если равно нулю, и –1, если отрицательно
`Sin(число)`	Синус числа
`sqr(число)`	Квадратный корень числа
`Tan(число)`	Тангенс числа

Производные математические функции

Если вы не находите в VBA ту математическую функцию, которая вам нужна, не отчаивайтесь — можно сконструировать выражение или создать процедуру типа `Function`, которая решит нужную задачу.

Если вы знаете математику настолько, что можете сами разобраться с тригонометрическими функциями типа обратной к функции косеканса, мне, наверное, не нужно объяснять вам, как использовать тригонометрические функции. Однако вам будет не лишним знать, что в справке VBA есть нечто типа шпаргалки по таким функциям, в которой вы найдете и выражения для их вычисления. Чтобы добраться до этой шпаргалки, поищите раздел *derived math functions* (производные математические функции) в содержании справки. Этот раздел справки напомнит вам, например, что логарифм числа X по основанию N вычисляется как `Log(X)/Log(N)`.

Если вы предполагаете, что какая-то производная математическая функция понадобится в дальнейшем, сделайте ее частью своего математического арсенала, записав в виде процедуры типа `Function`, и сохраните в модуле с названием вроде `МоиМатематическиеФункции`. Тогда вы сможете при случае воспользоваться ею, как только в этом возникнет необходимость.

Округление чисел

Округлять десятичные числа приходится часто, особенно при работе с денежными значениями. Как ни странно, VBA не предлагает прямого решения таких задач, но обсуждаемые ниже приемы помогут вам решить любые проблемы с округлением.

В VBA 6 есть функция `Round` (в VBA 5 такой функции нет), но дает она не слишком надежные результаты. Например, в результате выполнения оператора

```
x = Round(2.505, 2)
```

значением переменной x будет `2,5`, а не `2,51`, как должно быть. Поэтому не используйте `Round`, если, конечно, вы не намереваетесь получить заведомо неправильные ответы.

Есть другой простой способ округления чисел в VBA, при котором используется функция Format: просто используйте эту функцию, задав подходящее значение аргумента формат, как показано в следующем примере (как использовать аргумент формат, см. выше в разделе "Создание собственных форматов"):

```
sngОкругленное = Format(sngНеокругленное, "#,##0.00")
```

В VBA 6 для получения подобного результата можно использовать функцию FormatNumber:

```
sngОкругленное = FormatNumber(sngНеокругленное, 2)
```

В обоих этих примерах значение sngНеокругленное округляется до двух знаков после запятой. Чтобы при округлении с помощью функции Format получить иное число знаков после запятой, измените число нулей после десятичной точки в аргументе формат. Например, значение "#,##0.0" для аргумента в результате даст округление до одного знака после запятой. При округлении с помощью функции FormatNumber просто измените число, задающее значение второго аргумента, на нужное.

Не забывайте, что переменная, в которую вы помещаете округленное значение, должна иметь тип String, Single, Double, Decimal, Currency или Variant, но никак не тип Integer или Long, поскольку тогда вы просто потеряете дробную часть числа.

Предположим теперь, что нужно округлить некоторое значение не в дробной части, а до целых определенного порядка (например, до сотен). В этом случае выражение немного усложняется, но нельзя сказать, чтобы слишком. Вот пример округления до сотен:

```
sngОкругленное = Format(sngНеокругленное / 100, "#,##0.") * 100
```

Обратите внимание на то, что в данном случае строка для аргумента формат не имеет нулей справа от десятичной точки. Кроме того, неокругленное значение внутри функции Format делится на подходящую степень 10, а возвращаемый функцией результат умножается на ту же степень 10. Степень 10 должна быть равной порядку, до которого предполагается округление. В данном примере порядок равен 2, т.е. выполняется округление до сотен.

Работа со случайными числами

Если жизненная рутина угнетает ваш бурлящий дух, оживите свои программы определенной непредсказуемостью. Генерируемые случайным образом числа бывают нужны при создании программ, которые моделируют реальные процессы, при создании учебных программ и игр.

В VBA есть две встроенные команды для генерирования случайных чисел.

- ✓ Оператор Randomize. Используйте его, чтобы запустить генератор случайных чисел VBA. Не требуя аргументов, оператор Randomize запускает генератор, подавая ему на вход текущее системное время. Таким образом, гарантируется при каждом новом запуске новая последовательность случайных чисел.
- ✓ функция Rnd, Эта функция и поставляет случайные числа для использования в программе. Функция Rnd не имеет аргументов— вам нужно просто присвоить имя функции переменной или использовать Rnd в выражении. Возвращается значение с плавающей запятой (типа Single).

Числа с плавающей запятой часто используются в статистических и научных отчетах. Однако в некоторых ситуациях нужны случайные целые значения, если, например, необходима процедура, выбирающая случайным образом номер выигрышного билета. Чтобы преобразовать возвращаемое функцией Rnd значение в целое из определенного диапазона, используйте следующую формулу:

```
случайное_целое= Int(минимум +(Rnd() * максимум))
```

В этой формуле *максимум* и *минимум* задают соответственно верхнюю и нижнюю границы для значений получаемых случайных чисел — если границы заданы правильно, эта формула всегда будет давать число из соответствующего диапазона.

Финансовые функции

VBA предлагает ряд функций для подсчета рублей-копеечек в выплатах по ссуде или возвращениях инвестиций. *Я* не имею здесь возможности обсуждать все эти функции, но собираюсь все же рассмотреть функцию `Pmt` для расчета выплат по ссуде.

Расчет выплат по ссуде или процентов по вкладу

Используйте функцию `Pmt` для расчета сумм, которые предстоит выплачивать вам (или которые должны выплачивать вам) по закладной с фиксированными процентами или какому-либо другому займу. Вот формальный синтаксис функции Pmt:

```
Pmt(ставка, кпер, сумма[, цель[, тип]])
```

Первый аргумент, `ставка`, задает ставку процента выплат по займу *за определенный период* времени. Задаваемая вами ставка процента должна соответствовать периодичности выплат. Если нужно выплачивать ежемесячно по 8%-ной закладной, то не забудьте, что 8% является годовой нормой. В таком случае `ставка` должна задаваться выражением типа `.08/12` (8% разделенные на 12 месяцев).

Аргумент `кпер` должен задаваться целым значением, представляющим общее число выплат, которые предстоит сделать по займу. Для ежемесячных выплат в течение 5 лет `кпер` должно быть равно 5 * 12, или 60. Общая сумма займа задается аргументом `сумма`.

Остальные аргументы необязательны. Чтобы рассчитать, сколько вам нужно откладывать, чтобы накопить определенную сумму, задайте аргумент `цель`, чтобы передать функции Pmt запланированную сумму в виде *отрицательного* числа — это сумма на будущее. (При этом аргумент `ставка` задает ожидаемую ставку процента. Если некоторая часть из целевой суммы уже накоплена, эта часть должна задаваться аргументом `сумма`.) Аргумент `тип` позволяет указать, когда должны проводиться выплаты — в конце каждого периода (задается значением 0 и подразумевается по умолчанию) или в начале (значение 1).

Чтобы использовать функцию Pmt в программном коде, присвойте ее значение переменной типа Double. Например:

```
dblPay = Pmt (.08/12, 360, 300000) ' Типичная закладная
dblSav = Pmt (07/12, 120, 12500, -75000) ' Цель = 75000р.
```

Рог изобилия финансовых функций

Все разнообразие финансовых функций VBA представлено в табл. 11.8. Чтобы таблица была удобной для использования, я удалил из списков аргументов необязательные. Чтобы получить дополнительную информацию о любой из этих функций, обратитесь к соответствующему разделу справки VBA, где вы найдете также и подходящий пример.

Таблица 11.8. Финансовые функции VBA

Функция	Возвращаемое значение
DDB(стоимость, остаток, период)	Амортизация имущества за данный период, используя метод двойного процента со снижающегося остатка
FV(ставка, кпер, плата, сумма)	Будущее значение вклада на основе периодических постоянных платежей и постоянной процентной ставки
Impt(ставка, период, кпер, сумма)	Величина выплаты за указанный период на основе периодических постоянных платежей и постоянной процентной ставки
IRR(значения())	Внутренняя скорость оборота для ряда последовательных операций с наличными, представленными числовыми значениями
MIRR(значения(), фин_ставка, реинвест_ставка)	Модифицированная внутренняя скорость оборота средств для ряда последовательных периодических операций с наличными
Nper(ставка, платеж, сумма)	Общее количество периодов выплаты для данного вклада на основе периодических постоянных выплат и постоянной процентной ставки
NPV(ставка, значения())	Чистый текущий объем вклада, используя учетную ставку, а также объемы будущих платежей (отрицательные значения) и поступлений (положительные значения)
Pmt(ставка, кпер, сумма)	Величина выплаты по ссуде на основе постоянных выплат и постоянной процентной ставки
PPmt()	Платежи по процентам за данный период на основе периодических постоянных выплат и постоянной процентной ставки
PV (ставка, кпер, плата)	Текущий объем вклада на основе периодических постоянных платежей и постоянной процентной ставки
Rate(кпер, плата, сумма)	Процентная ставка за один период при выплате ренты
SLN(стоимость, остаток, период)	Величина непосредственной амортизации имущества за один период
SYD(стоимость, остаток, жизнь, период)	Годовая амортизация имущества для указанного периода

Другие встроенные команды

Даже после всех приведенных выше таблиц и обсуждений, обзор встроенных команд VBA остается далеко не полным. Некоторые из команд будут обсуждаться в следующих главах, но эту главу я завершу еще одной таблицей со множеством команд, которые вполне могут вам пригодиться.

Таблица 11.9. Различные встроенные команды VBA

При задании аргументов не забывайте заключать буквальные значения таких параметров, как путь, в кавычки.

Имя	Функция или оператор	Выполняемые действия (для операторов) и возвращаемые значения (для функций)
Работа с файлами		
ChDir путь)	Оператор	Изменяет текущий каталог (папку), но не активный диск
ChDrive(имя_диска)	Оператор	Изменяет активный диск
CurDir	Функция	Текущий путь (диск и имя каталога) в виде строки
Dir(путь)	Функция	Имя первого файла или каталога, соответствующего аргументу путь, который может содержать подстановочные символы
FileCopy источник, цель	Оператор	Копирует дисковый файл источник в цель (может включать путь, имя файла или и то, и другое)
FileDateTime(путь)	Функция	Значение типа Datec датой и временем создания или последнего изменения файла, заданного аргументом путь
FileLen(путь)	Функция	Длина (в байтах) файла, заданного аргументом путь
GetAttr(путь)	Функция	Целое значение, представляющее атрибуты файла или папки, заданных аргументом путь
Kill путь	Оператор	Удаляет один или больше файлов с диска (аргумент путь может содержать подстановочные символы)
MkDir путь	Оператор	Создает новый каталог
Name старый_путь As новый_путь	Оператор	Переименовывает и/или перемещает дисковый файл, каталог или папку
RmDir путь	Оператор	Удаляет каталог

Продолжение табл. 11.9

Имя	Функция или оператор	Выполняемые действия (для операторов) и возвращаемые значения (для функций)
Работа с файлами		
SetAttr путь, атрибуты	Оператор	Устанавливает заданные атрибуты дисковому файлу. Аргумент атрибуты задает числовое значение, являющееся суммой устанавливаемых атрибутов. VBA обеспечивает именованные константы для каждого из атрибутов
Разное		
AppActivate название, ожидание	Оператор	Активизирует другое выполняемое приложение. Если необязательный аргумент ожидание имеет значение True, то VBA будет ждать, пока ваша программа не получит фокус ввода
Beep	Оператор	Извлекает звук с помощью громкоговорителя компьютера
Choose(индекс, вариант1, вариант2, ... варианты)	Функция	Значение объекта из списка аргументов вариант!, **вариант2,**... варианты, **определяемого аргументом** индекс **(индекс должен задавать число)**
DoEvents	Функция	Позволяет Windows обрабатывать другие события, пока выполняется ваша программа. В VBA эта функция всегда возвращает О, поэтому нет необходимости приписывать ее значение переменной - используйте ее как оператор
Environ (строка) Environ (число)	Функции	Содержимое переменной окружения, заданной строкой с именем или числом, определяющим позицию
RGB(красный, зеленый, синий)	Функция	Целое число, представляющее RGB-значение цвета, заданного компонентами красный, зеленый, синий (используйте это значение для установки цветовых свойств объектов)
Randomize	Оператор	Инициализирует генератор случайных чисел
Rnd(число)	Функция	Случайное значение; аргумент число необязателен

Продолжение табл. 11.9

Имя	Функция или оператор	Выполняемые действия (для операторов) и возвращаемые значения (для функций)
Разное		
SendKeys строка, ожидание	Оператор	Пересылает заданную строкой последовательность символов в активное окно, как будто эти символы введены с клавиатуры. Если необязательный аргумент ожидание имеет значение True, то выполнение программы не продолжится, пока последовательность символов не будет обработана
Работа с реестром Windows		
Shell(путь)	Функция	Пытается выполнить программу, заданную аргументом путь. В случае успеха возвращает число, представляющее идентификатор программы, в противном случае возвращает 0
DeleteSetting имя_приложения, раздел, параметр	Оператор	Удаляет элемент реестра
GetAllSettings (имя_приложения, раздел)	Функция	Значение типа Variant, содержащее список всех установок для указанного раздела приложения в реестре Windows (в виде двумерного массива)
GetSetting (имя_приложения, раздел, параметр)	Функция	Значение параметра из указанного раздела приложения в реестре Windows
SaveSetting имя_приложения, раздел, параметр, значение	Оператор	Сохраняет значение в реестре
Форматирование вывода		
Spc(число)	Функция	Не возвращает никаких полезных значений, используется для размещения указанного числа пробелов в потоке вывода оператора Print # или метода Debug.Print
Tab(столбец)	Функция	Не возвращает никаких полезных значений, смещает поток вывода оператора Print # или метода Debug.

Окончание табл. 11.9

Имя	Функция или оператор	Выполняемые действия (для операторов) и возвращаемые значения (для функций)
Работа с переменными		

`IsDate (переменная)`
`IsNumeric(переменная)`
`IsObject(переменная)`
`IsArray(переменная)`
`IsNull(переменная)`
`IsEmpty(переменная)`
	Функции	`True`, если `переменная` имеет соответствующий тип или содержит значение соответствующего типа; иначе `False`

`Len (переменная)`	Функция	Число байтов, необходимое для хранения информации, помещенной в переменную
`TypeName (переменная)`	Функция	Строка, представляющая тип переменной
`VarType (переменная)`	Функция	Целое число, представляющее подчиненный тип переменной

Глава 12

Объектно-ориентированное программирование

В этой главе...

- Концептуализация объектов
- Понимание свойств, методов и событий — главных компонентов VBA-объектов
- Работа с объектными моделями
- Использование форм как объектов
- Выяснение и установка свойств объектов
- Вызов методов
- Использование ссылок на объект для идентификации объекта, с которым нужно работать
- Присваивание переменным ссылок на объект
- Создание своих собственных объектов с помощью модулей классов
- Эффективное использование объектов с помощью операторов `For Each ... Next` и `With`

Главным преимуществом VBA является статус объектно-ориентированного средства разработки приложений. Понимание объектов лежит в основе программирования в VBA, особенно когда дело касается создания пользовательских диалоговых окон и использования возможностей ведущего VBA-приложения. В этой главе мы сначала очертим объектное поле на уровне понятий; а затем займемся приемами программирования с использованием объектов. Объекты стали неотъемлемыми элементами ландшафта VBA. Именно через объекты вы получаете доступ к функциональным возможностям того VBA-приложения, в котором работаете. Точно так же вы можете получить доступ к объектам других VBA-приложений и даже создать свои собственные объекты.

Что такое объект

Можно, конечно, привести формальное определение объекта в VBA, но, я думаю, проще объяснить понятие объекта с помощью нескольких неформальных примеров, используя их функциональные возможности.

Объекты как компоненты VBA-приложений

Начать изучение объектов лучше всего с рассмотрения их как частей VBA-приложения и его документов. Любой элемент графики (shape) в Visio является объек-

том, равно как и любая *связь* (connect), установленная между двумя такими элементами графики (рис. 12.1). Точно так же объектами являются и слои, на которых размещаются элементы графики, и страницы, на которых располагаются слои. Объектом будет и сам документ, содержащий все эти страницы, слои, элементы графики и связи.

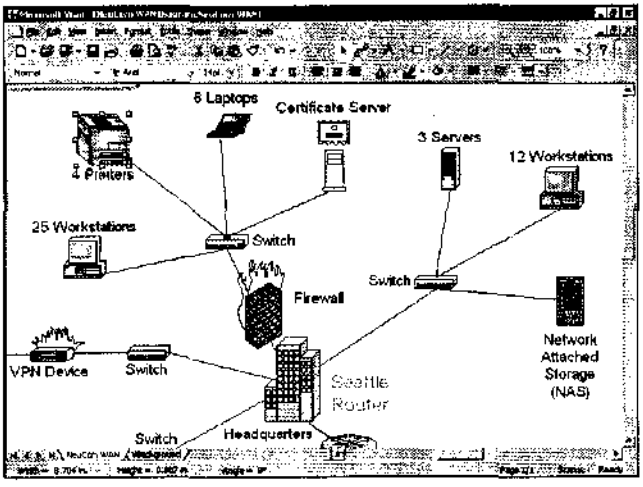


Рис. 12.1Примерыобъектов VBA

Подобным образом к объектам Excel относятся ячейки, в которых размещаются данные и формулы, именованные диапазоны ячеек, диаграммы, украшающие многие документы, отдельные рабочие листы и целые рабочие книги. И во всех приложениях Microsoft Office — как и во многих других VBA-приложениях — меню и панели инструментов, а также пункты этих меню и кнопки в этих панелях инструментов тоже являются объектами.

Объекты VBA существуют в иерархии, в которой объекты одних типов содержатся в объектах других типов. Подобные иерархии объектов рассматриваются ниже, в разделе "Что такое объектная модель", а пока что мы сконцентрируем внимание на обсуждении понятия индивидуального объекта.

Объекты на уровне понятий

Если вам трудно ассоциировать объекты с графическими элементами, ячейками рабочего листа или кнопками панели инструментов, представляйте себе объекты как *материальные предметы*. Вы же можете представить, как вырезаете ножницами из листа бумаги кружочек и наклеиваете его на другой лист бумаги. А в ячейку рабочего листа вы можете впечатать числа, на кнопке можете щелкнуть.

Правда, кроме таких конкретных объектов, VBA-приложения предлагают и более абстрактные.

- ✓ В Microsoft Excel объект `CustomView` задает пользовательское представление рабочей книги (в Excel пользовательское представление определяет внешний вил рабочей книги на экране и ее параметры печати).
- ✓ В Microsoft Word объект `FileSearch` представляет, как сказано в соответствующем разделе справки, "функциональные возможности диалогового окна Открытие документа (меню Файл)". Обратите внимание, этот объект представляет не само диалоговое окно, а его функциональные возможности.

- ✓ В Visio объект Style представляет атрибуты линий, текста и заливки для графических элементов.
- ✓ В VBA имеется несколько объектов, доступных для всех VBA-приложений. Так, например, объект Collection представляет контейнер для переменных или других объектов, с которыми, независимо от типов этих объектов, предполагается работать как с единым целым.

Практическое определение объекта

Вы можете заметить, что порой представить объект VBA в материально осязаемом виде не так-то просто. Но это и к лучшему — чем дальше вы уходите от материализации объектов, тем свободнее будете себя чувствовать при работе со всем диапазоном доступных объектов. Используемое программистами-практиками определение объекта оказывается совсем простым: объектом называется любая именованная сущность, имеющая

- ✓ свойства, т.е. установки, которые можно проверить и изменить;
- ✓ методы, т.е. действия, которые может выполнить объект, когда программа попросит об этом;
- ✓ события, т.е. ситуации, в которых объект оказывается и на которые может ответить заранее определенными для таких ситуаций действиями.

Если вам не чуждо чувство прекрасного, вы можете заметить, что для таких щедро одаренных созданий термин *объект* не очень-то подходит, поскольку эти объекты больше похожи на братьев наших меньших, чем на инертные лампы. У тигров и китов есть отличительные особенности глаз, конечностей и хвостов, а у объектов — свойства. Лошади и собаки умеют по команде выполнять разные трюки или убегать от опасности, а объекты имеют методы и события.

Вы все еще хотите услышать формальное определение? Вот оно: объектом в программе называется именованная единица, объединяющая в себе данные и программный код, использующий эти данные и воздействующий на них. В таком случае говорят, что объект *инкапсулирует* данные и связанный с ними программный код.

Классы объектов и конкретные объекты

Есть одно формальное обстоятельство, которое следует запомнить. Существует различие между конкретными объектами и теми образцами, на основе которых данные объекты создаются. Конкретный объект представляет отдельный документ, графический элемент, ячейку рабочего листа или нечто другое. Например, документ, как объект, представляет текст этого отдельного документа.

В противоположность конкретному объекту, *класс объекта* можно сравнить со строительным проектом. По одному проекту можно построить множество домов, в которых будут жить люди, но в самом проекте никто не живет. Точно так же в классе планируются типы данных, которые могут использоваться объектом, и определяются методы, свойства и события объекта. На основе этого плана вы создаете *экземпляр* данного класса — тот объект, который будет содержать реальные данные. На основе одного класса можно создать столько объектов, сколько вы пожелаете, чтобы использовать каждый из них для различных целей и с различными данными.

Коллекции объектов

Коллекция — это VBA-объект специального назначения. Как видно из самого названия, коллекции предназначены для упрощения работы с набором объектов, когда этот набор объектов нужно использовать как одно целое. Как правило, все объекты в коллекции имеют один и тот же тип. Например, коллекция Shapes в Visio состоит из объектов Shape, а коллекция Pages — из объектов Page.

Но есть и коллекции, которые оказываются более либеральными к типу входящих в них объектов. Например, VBA предлагает родовой объект Collection, предназначенный для хранения в нем объектов любых типов в любой комбинации. Если возникнет необходимость готовить самые невероятные смеси, то коллекции VBA окажутся как раз кстати.

Позже, в разделе "Работа с коллекциями объектов", мы с вами выясним, как получить доступ к отдельным объектам в коллекциях, а в главе 13 вы узнаете, как использовать свойства и методы самих коллекций (не забывайте, что они тоже являются объектами) и как создавать свои собственные коллекции на основе родового VBA-объекта Collection,

Что такое объектная модель

Как уже упоминалось, объекты VBA существуют в иерархической зависимости один от другого. Кроме обладания своими собственными свойствами, методами и событиями, объекты высших ступеней иерархии служат в качестве *контейнеров* для одного или целого множества подчиненных объектов. Эти вложенные объекты в свою очередь содержат другие объекты и т.д.

Вся система таких иерархических отношений конкретного VBA-приложения называется *объектной моделью* этого приложения. Часто представляемая в графическом виде, объектная модель приложения описывает вложения одних объектов в другие. Пример графического представления одной из объектных моделей показан на рис. 12.2.

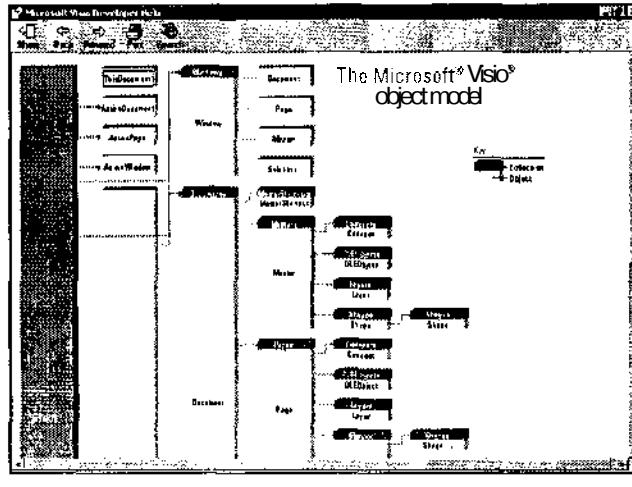


Рис. 12.2. Объектная модель Visio с компакт-диска Visio 2000

Как видно из рис. 12.2, на вершине объектной модели VBA-приложения находится *объект Application* (приложение), являющийся контейнером для всех других объектов приложения, с которыми вы можете работать. Ваш VBA-*проект* тоже является объектом-контейнером, в котором размещаются все созданные вами *модули* программного кода и формы, равно как и документ проекта. (О структуре VBA-проекта и его компонентах говорилось в главе 7.)

Понимание важности объектной модели

Вам придется сообщать VBA о том, какой конкретный объект нужен вам для работы, поэтому хорошее понимание объектной модели VBA-приложения оказывается очень важным для эффективной работы в нем. С помощью диаграммы объектной модели приложения, подобной показанной на рис. 12.2, очень просто проследить всю цепочку объектов, которой принадлежит нужный вам объект. В данном случае, например, сразу видно, что объект Connect является членом коллекции Connects, принадлежащего объекту Master, который в свою очередь... .

Расширение объектной модели

В программах, созданных в VBA, совсем не обязательно ограничивать себя использованием объектов только одного VBA-приложения. Не обязательно даже ограничиваться использованием только VBA-приложений как таковых. Можно использовать вообще любые приложения и "компоненты", удовлетворяющие стандарту *Component Object Model* (COM) фирмы *Microsoft* (Component Object Model можно перевести как *объектная модель компонента).*

Стандарт COM представляет собой набор технических спецификаций, регламентирующих правила определения объектов в приложениях и других программных единицах, а также правила *предоставления* таких объектов для использования в других приложениях. Чтобы обозначить, что удовлетворяющим стандарт COM приложением можно управлять из другого приложения, используется специальный термин *автоматизация.* Кстати, стандарт COM относится не только к VBA или Visual Basic, он понятен многим другим средствам разработки приложений (например, компиляторам C++), которые таким образом могут использовать соответствующие объекты.

Стандарт COM открывает фантастические возможности для создания мощных VBA-приложений, допускающих настройку пользователем. Теперь вы можете (я не сказал легко) создавать приложения, позволяющие одновременно обрабатывать информацию, полученную из диаграмм Visio, документов Word, рабочих листов Excel и т.п. Ваше приложение может отображать всю эту информацию в окнах, которые разработаны вами, но которые в то же время используют специальные функциональные возможности отображения информации этих приложений-компонентов.

Разработка пользовательских приложений такого уровня, очевидно, относится к непростым задачам, поэтому в данной книге это будет обсуждаться весьма поверхностно (в главе 14). Но я должен по крайней мере упомянуть, что возможности VBA открывают путь к великим свершениям.

Формы в VBA — это тоже объекты

Формой называют любое созданное в VBA пользовательское окно. Если в вашей программе предусмотрен ее собственный пользовательский интерфейс, формами будут как главное окно программы, так и все другие диалоговые окна, отображаемые вашей программой (рис. 12.3).

Главное, здесь нужно понять, что формы VBA сами являются объектами, т.е. представляют собой сущности, содержащие как информацию (задающую, например, внешний вид формы на экране), так и набор средств для обработки этой информации. Официально формы в VBA описываются в терминах объекта `UserForm`.

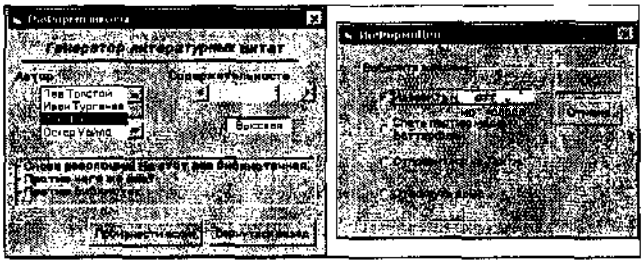


Рис.12.3. Примеры форм VBA

Как и любые другие уважающие себя объекты в VBA, формы прекрасно вписываются в рамки парадигмы объектной модели. Каждый объект `UserForm` принадлежит одновременно двум коллекциям объектов — VBA-проекту, в котором хранится форма, и коллекции `UserForms`, содержащей все формы, загружаемые программой. В свою очередь, объект `UserForms` служит контейнером для коллекции `Controls`, содержащей элементы управления, помещенные в форму.

Точно так же элементы управления в форме — кнопки, флажки, переключатели и всевозможные другие рычажки, которыми можно любоваться и играть, — тоже являются объектами. Для каждого типа элементов управления в VBA предлагается объект соответствующего типа.

Техника, с помощью которой можно получить доступ к конкретной форме или помещенному в нее элементу управления и использовать свойства, методы и события этого объекта, ничем не отличается от техники, используемой с другими объектами, о которых говорится в этой главе. Подробнее о работе с формами и элементами управления (а значит, и соответствующими объектами) говорится в главах 10 и 19.

Использование объектов в программе

Теперь, когда теоретические основы пройдены, наступает время практических рекомендаций по программированию с использованием объектов VBA. Начнем, пожалуй, с использования в программном коде свойств, методов и событий объектов, а уж после этого перейдем к более тонким вопросам идентификации объектов, которые предполагается использовать.

Основные правила программирования объектов

Хотя в самом понятии объекта и есть кое-что, над чем можно поломать голову, *использовать* объекты достаточно просто. Поскольку объекты имеют имена, вы всегда можете знать, с каким из объектов работаете. Чтобы идентифицировать в программном коде метод или свойство объекта, с которыми предполагается иметь дело, нужно просто напечатать имя объекта, за ним точку и, наконец, имя метода или свойства.

Например, `MyShape.LineStyle` идентифицирует свойство `LineStyle` объекта с именем `MyShape` в Visio. Точно так же `MyWorksheet.Calculate` идентифицирует метод `Calculate` объекта рабочего листа с именем `MyWorksheet` в Excel.

Правда, здесь вы можете задать себе вопрос: "Но как же мне узнать имя объекта, которое необходимо поместить в начале?" Блестящий вопрос! Вот только ответ на этот вопрос требует достаточно пространных пояснений, которые я отложу до раздела "Идентификация объекта для использования", сосредоточившись сначала на обсуждении свойств, методов и событий. А пока что примите на веру, что объект можно вызвать по его имени.

Использование объектных моделей

Как вам стало очевидным из раздела "Что такое объектная модель", каждое VBA-приложение имеет свою неповторимую *объектную модель* (иерархию объектов), а каждый из его объектов имеет свой собственный набор свойств, методов и событий. Чтобы заставить эти объекты работать в программе, вам необходимо хорошо разобраться в деталях объектной модели соответствующего приложения.

Во многих приложениях справка о VBA содержит специальный раздел, в котором вы найдете фафическое представление объектной модели этого приложения (рис. 12.4). Если диаграмма объектной модели не появилась перед вами при первом же вызове справки, перейдите в окне справки на вкладку Содержание, где она будет представлена под рубрикой типа Microsoft Excel Objects. Используя диаграмму объектной модели в качестве гида, вы сможете открыть информацию о конкретном объекте, дважды щелкнув на его имени в диафамме. Иначе получить нужную информацию можно, отыскав в указателе справки соответствующий раздел и открыв его. Как правило, в любом разделе, посвященном объекту, имеются ссылки на другие разделы, содержащие дополнительную информацию о свойствах, методах и событиях соответствующего объекта. (Заметьте, здесь идет речь о справке о VBA для соответствующего приложения, а не об общей справке о приложении или VBA. О работе со справочной системой VBA см. главу 4.)

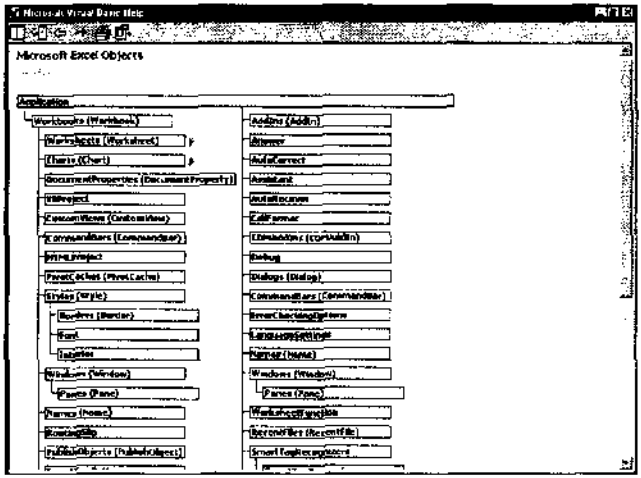


Рис. 12.4. Объектная модель Excel

Другим жизненно важным средством для изучения свойств, методов и событий объектов и их взаимосвязей в приложении является обозреватель объектов в редакторе Visual Basic. Полное описание обозревателя объектов см. в главе 6.

Выяснение и установка свойств объектов

Вероятно, свойства — это самые простые для понимания характеристики объекта. Каждое *свойство* хранит информацию о некотором аспекте внешнего вида, поведения, содержимого объекта или, если хотите, о его генеалогии.

Например, объект Document может иметь свойство Pages, говорящее о числе страниц в документе. Объект Shape может иметь свойство Fill, задающее цвет графического эле-

мента. Объекты CommandButton (представляющие кнопки в диалоговых окнах) имеют свойство Caption, задающее текст, появляющийся на кнопке.

Как правило, VBA-программа может выяснить, какая информация хранится в том или ином свойстве, чтобы затем принять решение о том, следует ли предпринять какие-либо действия или просто отобразить на экране текущие установки свойства. Программа может и, наоборот, изменить установки свойств с тем. чтобы соответствующий объект изменил свой внешний вид или поведение, — но только в том случае, когда свойство допускает такое изменение (многие свойства допускают чтение своих значений, но не их изменение).

Изо всех типов свойств легче всего разобраться со свойствами, задающими различные аспекты внешнего вида объекта. Например, объект Shape в Visio среди прочих свойств имеет следующие.

Свойство	Описание
AreaUI	Площадь графического элемента в специальных внутренних единицах
FillStyle	Стиль Visio, задающий цвет графического элемента и другие характеристики заливки
Text	Текст, появляющийся на графическом элементе

Эти свойства задают внешний вид объекта в документе.

Свойства, относящиеся к поведению объекта, определяют реакцию объекта на внешние раздражители. Например, элементы управления в формах являются объектами, среди свойств которых свойство Enabled определяет, будет ли соответствующий элемент управления вообще реагировать на такие события, как щелчок кнопки мыши на нем.

Некоторые свойства, например AreaUI для объекта Shape в Visio, могут принимать любое значение. Другим же допускается назначить только одно из некоторого списка заранее определенных значений. Многие свойства могут принимать только одно из двух возможных значений, таких как True или False. Горячий или Холодный, Сухой или Мокрый.

В любом случае, для выяснения текущего значения свойства или его изменения можно использовать простые операторы VBA.

Что нельзя делать с некоторыми свойствами

Знание того, *как* менять значение свойства, еще не означает, что его *можно* менять. Некоторые свойства позволяют выяснить свои значения, но не позволяют менять их. Это свойства *только для чтения.* Существуют также, хотя встречаются значительно реже, свойства *только для записи* — их значения можно изменить, но прочитать их текущие значения нельзя. Однако большинство свойств предназначено *для чтения и записи* — их значения можно и прочитать, и изменить.

Установки свойств — это данные

Главной задачей свойства является описание некоторой характеристики объекта, но вы должны знать, что установки свойства — это обычные данные, в принципе ничем не отличающиеся от тех данных, которые вы размещаете в переменных VBA. А если так, то можно представлять себе свойства как более или менее постоянно существующие переменные, которые не требуется объявлять.

При таком представлении переменных можно говорить, что любое свойство хранит данные определенного типа точно так же, как переменная. Например, свойство, которое может принимать только два взаимоисключающих значения (True или False. Счастливый или

Несчастный, Левый или Правый), имеет тип Boolean. Одни свойства являются строками, другие— целочисленными значениями, третьи — десятичными числами с дробной частью или плавающей запятой и т.д. Свойства могут быть и объектами. (О допустимых в VBA типах данных см. главу 7.)

Выяснение текущего значения свойства

Чтобы выяснить или, как говорится, *получить* текущее значение свойства, используйте свойство так, как будто это функция или процедура типа Function. А значит, присвойте это свойство некоторой переменной в своем программном коде. Переменная при этом должна иметь тот же тип, что и свойство, или, по крайней мере, иметь подходящий тип.

В следующем примере объект представляет собой, скажем, вопрос для вступительного экзамена. Свойство, значение которого выясняется здесь, описывает сложность вопроса по десятибалльной шкале;

```
Dim intСложность As Integer
intСложность = objЭкзаменационныйВопрос.УровеньСложности
```

Первый оператор объявляет переменную, которой будет присвоено текущее значение свойства, что и делает второй оператор.

Но для чего выяснять текущее значение свойства? Как правило, это значение используется в условных операторах и на его основе принимается решение о выполнении или невыполнении программой определенных действий. (В данном случае, например, это может быть "если УровеньСложности вопроса выше 8 и если ответ правильный, то количество баллов за ответ удвоить".) Значение свойства удобно сохранить в переменной и тогда, когда нужно присвоить это значение такому же свойству другого подобного объекта.

Правда, если значение свойства используется только однажды, совсем не обязательно присваивать его переменной — в выражении можно использовать и само свойство. Например:

```
If objЭкзаменационныйВопрос.УровеньСложности > 8 Then
     intОбщееЧислоБаллов = intОбщееЧислоБаллов + _
(intЧислоБалловЗаОтвет *2)
End If
```

Такая практика удобна, но не забывайте, что частое обращение к свойствам объектов замедляет работу программы. Если значение некоторого свойства используется больше одного-двух раз, это значение лучше сохранить в переменной — VBA извлекает значение обычной переменной значительно быстрее, чем значение свойства.

Изменение значения свойства

Не забывайте, что свойства — это просто обремененные славой переменные. Поэтому свойствам можно присваивать значения точно так же, как обычным переменным, поместив в строке имя свойства слева, а значение — справа от знака равенства. В операторе

```
objМузыкаМеталл.ФакторРаздражения = 999
```

значение свойства ФакторРаздражения — наверное, мера искажений, резонансов и всевозможных шумов — для объекта objМузыкаМеталл устанавливается равным 999:

```
objМузыкаМеталл.Мелодия = False
objМузыкаМеталл.Название = "У меня блохи. Плохо."
```

Свойства, выбираемые по умолчанию

Многие объекты имеют свойство, выбираемое по умолчанию. Прочитать или установить значение такого свойства можно, указывая только имя объекта и не указывая имя самого свойства. Обратившись снова к предыдущему примеру, предположим, что для объекта objМузыкаМеталл свойством по умолчанию является Название. Тогда последний оператор в примере можно записать проще:

```
objМузыкаМеталл = "У меня блохи. Плохо."
```

Использовать свойства по умолчанию удобно, конечно, когда вы точно знаете, какое именно свойство объекта выбирается по умолчанию. Если же вы сомневаетесь или предполагаете, что потом не удастся вспомнить, какое именно свойство выбирается по умолчанию, то лучше в таком случае напечатать имя свойства.

Объекты в роли свойств

Как уже упоминалось в разделе "Установки свойств — это данные", свойство одного объекта может указывать на другой объект. Это дает возможность обращаться к дочерним объектам данного *объекта-контейнера* так, как будто дочерние объекты являются свойствами этого контейнера. Например, в выражении

```
Toolbar.ToolbarItems
```

ToolbarItems — свойство объекта Toolbar, но значением этого свойства является объект ToolbarItems.

На самом деле именно такое использование свойств объектов является основным средством идентификации объектов для использования в программе (подробности приведены ниже, в разделе "Идентификация объекта для использования").

Вверх по родовому дереву

Подобно тому, как свойства объекта указывают на подчиненные объекты, с помощью свойств можно выяснить, какому контейнеру принадлежит сам объект. Например, если в Visio в переменной хранится ссылка на объект Pages и вы хотели бы выяснить, к какому документу она относится, то выражение

```
Pages.Parent
```

возвратит ссылку как раз на нужный документ. А если вам потребуется узнать, какому *приложению* принадлежит данный объект, то, как правило, можно пропустить все промежуточные ступени иерархии и обратиться сразу на самый верхний уровень с помощью свойства Application:

```
Pages.Application
```

Методы в действии

Методы — это именованные действия, которые объект может выполнить по команде. Ввиду того, что любой метод является неотъемлемой частью объекта, объект сам знает, что ему делать, когда вы вызываете метод.

Например, объект, задающий графический элемент, может иметь метод Resize, изменяющий размеры, и метод Rotate, поворачивающий графический элемент на странице документа. Объект, представляющий ячейку рабочего листа, может иметь метод Calculate, пересчитывающий значение в ячейке, и метод Clear, удаляющий ее содержимое. Объект, представляющий целый документ, скорее всего имеет методы Print (Печать) и Save (Сохранить).

На самом деле, методы представляют собой не что иное, как процедуры, привязанные к конкретному объекту. Чтобы вызвать метод, необходимо напечатать имя объекта, точку, а затем имя метода. Оставив в стороне сумасшествие металла, давайте рассмотрим объект `objМузыкаДжаз` в предположении, что он представляет цифровую запись джазовых композиций в мультимедиа-программе. Вполне возможно, что такой объект имеет метод `Воспроизведение`. Вот как его можно вызвать:

```
objМузыкаДжаз.Воспроизведение
```

В основном техника вызова методов вполне согласуется с техникой вызова процедур и функций VBA, о которой шла речь в главе 7.

- ✓ Чтобы вызвать метод, требующий ввода аргумента, напечатайте аргумент после имени метода. В следующем примере метод `Воспроизведение` получает аргумент, говорящий о том, сколько раз нужно воспроизвести пьесу:

```
objМузыкаДжаз.Воспроизведение 3
```

- ✓ Если метод имеет несколько аргументов, печатайте их через запятую (в следующем примере предполагается, что `Негромко` является именованной константой, задающей невысокий уровень громкости, а `intСкорость` — переменной, определяющей скорость воспроизведения):

```
objМузыкаДжаз.Воспроизведение 2, Негромко, intCKopocTb
```

- ✓ Чтобы вызвать метод, возвращающий значение, присвойте метод переменной или используйте его в выражении как обычную процедуру типа `Function`. В этом случае все аргументы, независимо от их числа, необходимо заключить в скобки:

```
dateВремяВоспроизведения =
_objМузыкаДжаз.Воспроизведение(2, Негроко,intСкорость)
```

Как и в случае свойств, методы совершенно разных классов объектов могут иметь одинаковые имена. Например, объекты, содержащие группы элементов или других объектов, как правило, имеют метод Add.

Обратите внимание, что метод меняет установки одного или нескольких свойств. Например, объект `objМузыкаДжаз` может иметь свойство только для чтения `ЧислоИсполнений`, допускающее изменение только методом `Воспроизведение`, но его значение в то же время можно выяснить оператором типа `intИсполнения = objМузыкаДжаз.ЧислоИсполнений`. Некоторые объекты имеют методы, единственное назначение которых — установка значений определенных свойств.

События

Событие представляет собой нечто, случающееся с объектом, и на что объект может ответить заранее предусмотренным действием. К событиям можно отнести следующее.

- ✓ Физические действия пользователя программы, например щелчок кнопкой мыши, перемещение курсора, прыжки и пляски вокруг компьютера. (Ну, ладно-ладно, прыжки и пляски вокруг компьютера, очевидно, не относятся к событиям, распознаваемым VBA.)

- ✓ Ситуации, в которые попадает объект в ходе выполнения программы. Например, если речь идет об объекте, представляющем документ, то к событиям можно отнести открытие и закрытие документа, добавление и удаление страниц (такие события в Word распознаются объектами `Application` и `Document`). В Visio объекты `Pages` могут отвечать надевять различных событий, среди которых `BeforeShapeDelete` (Перед удалением графического элемента) и `TextChanged` (Изменение текста). Другие примеры таких не связанных с формами событий вы найдете в главе 4.

То, какие события данный объект может распознавать, определяете не вы, а ваше VBA-приложение. К *вашим* задачам относится создание программного кода, определяющего, что должен делать объект в ответ на распознаваемое событие. Здесь следует обратить внимание на такой момент: вам не нужно вызывать процедуру обработки события из программного кода — при наступлении соответствующего события объект автоматически предпримет запрограммированные вами действия.

Программирование процедур обработки событий для форм рассматривается в главе 10. Те же самые приемы, в основном, применяются и в случае не связанных с формами событий, правда, тут существуют некоторые тонкости, варьирующие от приложения к приложению, а иногда даже в рамках одного приложения. Исчерпывающая информация по этому вопросу должна содержаться в справочной документации вашего приложения или в файлах его справки.

использования

Вы должны сообщить VBA, какой именно объект нужен. Для этого используется *объектное выражение* —VBA-выражение специального вида, которым однозначно определяется конкретный объект для использования. То остающееся за кулисами значение, которое VBA вычисляет на основе объектного выражения, является *ссылкой на объект* (вы можете представлять себе такое значение как почтовый адрес. по которому проживает объект).

Если при работе с объектами объектное выражение составлено правильно — цель практически достигнута. После этого можно значительно упростить себе жизнь, создав для объекта переменную и присвоив этой переменной ссылку на объект, используя построенное объектное выражение. С этого момента в программном коде можно ссылаться на объект просто по имени переменной.

Понимание объектных выражений

Объектное выражение представляет собой *фрагмент программного кода* — выражение, "указывающее" на конкретный объект. С помощью правильно построенного объектного выражения можно изменять свойства объекта, вызывать его методы или присвоить объект переменной.

Обсуждаемые здесь приемы исключительно важны для практического программирования с использованием VBA, хотя сразу понять их не очень просто. Из-за того что в программе могут использоваться много объектов одновременно, полное объектное выражение должно включать все объекты, содержащие нужный вам обьект. Попробую пояснить это следующим образом. Предположим, вас просят отправиться и найти "того парня". Вы сразу же спросите: "Какого еще того?" А вот если бы вас попро-

сили найти самого старшего школьника, живущего в доме №3 по улице Цветочной в городе Шепетовка, Украина, то вам не пришлось бы задавать такой вопрос. (Правда, вы бы тогда, наверное, спросили "Зачем?"') Конечно, если вы уже находитесь в доме №3 по улице Цветочной, и в том доме живет только один школьник, то команда типа "покормить мальчишку" будет вполне адекватной. Точно так же, когда контекст ясен, и VBA не требует полного списка объектов.

Работа со свойствами, являющимися объектами

Свойство одного объекта может тоже оказаться объектом. Объекты сосуществуют в иерархии, когда один объект (например, документ) служит контейнером для других, подчиненных ему объектов (страниц, рабочих листов или чего-то другого).

Связь между двумя этими идеями очевидна: если объект содержит подчиненные объекты, то любой подчиненный объект можно идентифицировать посредством свойства первого объекта. Выражение, которое используется для указания нужного свойства, *является* объектным выражением.

Например, рассмотрим следующее выражение, идентифицирующее конкретный объект в документе Word;

```
ThisDocument.Sections(2).Range
```

В этом выражении не одна, а две точки. Посмотрите, Range является свойством объекта Section, который, в свою очередь, является свойством объекта ThisDocument. Возможно, чтобы понять идею, вам этого примера уже достаточно, но если нет, то в следующем разделе вы найдете более подробные объяснения.

Получение объектов

В большинстве объектных моделей VBA объект Application (Приложение) находится на самой вершине иерархии. Однако обычно не возникает необходимости включать этот объект в выражения — VBA соображает достаточно неплохо для того, чтобы понять, что вы работаете с объектами текущего приложения, если вы специально не указываете иного.

Объект Application содержит объект Documents — коллекция, представляющая все открытые документы. Если вам нужно работать с конкретным документом, необходимо идентифицировать его среди членов этого объекта-коллекции Documents. Например, Documents (5) представляет объект Document № 5 в коллекции Documents.

В примере объектного выражения вы, однако, видите ключевое слово ThisDocument. Во многих VBA-приложениях ThisDocument используется для обозначения объекта Document, ассоциированного с вашим проектом.

В Word каждый объект Document имеет свойство Sections, которое ссылается на коллекцию Sections,— объект, представляющий множество всех разделов документа. Поэтому первая часть указанного выше выражения — ThisDocument.Sections — идентифицирует конкретную коллекцию Sections, принадлежащую объекту ThisDocument. Идентифицировав объект Sections, вы получаете возможность выбрать конкретный член этой коллекции. Таким образом, Sections(2) ссылается на второй раздел документа.

Теперь о завершающей части выражения. Тонкость здесь заключается в том, что хотя .Range идентифицирует *свойство* объекта Section, значением этого свойства является *объект* Range. Таким образом, все выражение обеспечивает ссылку на этот объект Range. Использование выражения, подобным образом прокладывающего путь к конкретному объекту сквозь дебри иерархии объектов, в профессиональной фразеологии VBA называется "получением объекта".

Работа с коллекциями объектов

В рассмотренном выше примере объектного выражения

```
ThisDocument.Sections(2).Range
```

объект `Sections` представляет собой *коллеекцию*. В VBA коллеекцией называется объект специального типа, который может содержать любое число других объектов.

Определенные в VBA-приложениях коллекции обычно содержат объекты одного типа. Например, коллекции `Sections` в Word содержат только объекты `Section`, а коллекции `Pages` в Visio могут принадлежать только объекты `Page`. (Правда, VBA предлагает и родовой объект `Collection`, в котором вы можете разместить объекты любых типов. О том, как создавать и использовать свои собственные коллекции такого рода, говорится в главе 13.)

О коллекциях вам достаточно знать только то, как получить доступ к содержащимся в них конкретным объектам. Для этого у вас есть две возможности.

- ✓ Сослаться на объект по его месту или *номеру индекса* в коллекции. Именно это и сделано в рассмотренном выше примере объектного выражения — `(2)` задает ссылку на второй объект `Section` в коллекции `Sections`.
- ✓ Сослаться на объект по имени. Многие объекты имеют имена. Если вам известно имя объекта, который необходимо использовать, то вместо номера в объектном выражении можно указать имя. Например, в Visio объекты `Page` могут иметь имена, поэтому вполне допустим следующий пример:

```
ThisDocument.Pages ("Моя любимая страница")
```

Создание переменных для объектов

Никто не любит вводить длинные и сложные объектные выражения, даже если они достаточно понятны, как выражение `ThisDocument.Sections(2).Range`. Если в программе один и тот же объект используется несколько раз, создайте для него переменную, в которой будет храниться ссылка на объект. Тогда, при необходимости указать объект, вместо полного объектного выражения вы сможете ввести имя переменной.

Обычно *объектная переменная* имеет более короткое имя, которое легче запомнить и напечатать, чем оригинальное объектное выражение; кроме того, она имеет еще пару преимуществ. Во-первых, использование объектной переменной ускоряет выполнение программного кода, поскольку в этом случае VBA может обратиться к объекту напрямую, а не пробираться сквозь частокол свойств ряда промежуточных объектов. Во-вторых, одну и ту же переменную можно использовать для сохранения ссылок на различные объекты. Таким образом, программный код можно сделать более гибким, поскольку появляется возможность выбирать, ссылку на какой объект должна хранить переменная во время выполнения программы.

Процесс создание объектной переменной можно разбить на два этапа.

1. **Создание переменной, которая будет использоваться для хранения ссылки на объект.**
2. **Присваивание переменной ссылки на объект, с которым в дальнейшем необходимо будет работать.**

Подробно каждый из этих этапов обсуждается в следующих двух разделах.

Объявлениеобъектнойпеременной

Объектные переменные объявляются точно так же, как переменные любого другого типа. Стандартным при этом является использование оператора Dim (но точно так же, как и с переменными других типов, вы можете использовать в объявлении ключевые слова Public, Private или Static вместо Dim). Вот два примера объявления объектных переменных:

```
Dim objВеликийОбъект As Object ' типичный объект
Dim objShapeObject As Shape ' объект Shape
```

Существует одно важное различие между этими двумя объявлениями. В первом операторе объявляется объектная переменная, но не указывается тип объекта, который будет в ней храниться. Переменную такого типа можно использовать по мере необходимости для ссылок на объекты любого типа. Во втором операторе объявляется конкретный тип, или *класс* объекта, для которого предназначается создаваемая переменная. VBA не позволит поместить в такую переменную объекты иных классов.

Эти два типа объявлений объектов имеют специальные названия: *поздняя привязка*, когда конкретный класс не указывается, и *ранняя привязка,* когда переменной сразу же назначается определенный класс объектов.

Всегда, когда это возможно, лучше использовать раннюю привязку, объявляя класс объектной переменной. При ранней привязке вы получаете следующие преимущества,

- ✓ **Уменьшается вероятность появления ошибок.** Как уже упоминалось, в этом случае VBA не позволит присвоить переменной объект неправильного типа. Кроме класса объекта, компилятор проверит также и остальную часть программного кода, чтобы убедиться в том, что вы везде используете только методы и свойства, допустимые для объектов данного класса. При поздней привязке компилятор не смог бы выполнить такую проверку, и попытка использования недопустимого свойства или метода обнаружилась бы только во время выполнения программы.
- ✓ **Ускоряется выполнение программы.** Поскольку компилятор может определить, имеет ли объект те методы и свойства, которые вызываются в программе, самой программе не придется тратить время на проверку этого во время ее выполнения.
- V **Упрощается программный код.** Чтобы выяснить, для какого класса объектов предназначена данная переменная, вам нужно просто взглянуть на строку с ее объявлением.

Несмотря на все эти преимущества, поздняя привязка предпочтительнее в следующих двух ситуациях.

- ✓ Когда одну и ту же переменную *предполагается* использовать для объектов различных классов. Это оправданно, когда разным классам присуши одинаковые методы или свойства, используемые в программе. Имея подобную переменную, вам не придется переписывать программный код несколько раз, чтобы делать одно и то же с разными объектами.
- ✓ Когда объекты вызываются из других приложений и поэтому их нельзя объявить с помощью ранней привязки. Использование объектов из других приложений относится к углубленным вопросам VBA-программирования (краткое обсуждение которых вы найдете в главах 10 и 19), но вам не мешает знать о таком потенциальном ограничении.

Как вы знаете, переменная типа Variant может хранить информацию любого типа, в том числе и ссылки на объекты. Если вы объявляете переменную без явного указания на то, что это объект, то, выбирая позднюю привязку, вы сможете разместить в ней ссылку на объект позже.

Присваивание переменной ссылки на объект

После того как объектная переменная объявлена, перед ее использованием необходимо поместить в нее ссылку на объект. Это делается путем присваивания этой переменной объектного выражения с помощью ключевого слова Set. Например:

```
Set objShapeObject = ThisDocument.Pages(1).Shapes(4)
```

Обратите внимание, что синтаксис в данном случае немного отличается от того, каким вы присваивали данные обычным переменным (см. главу 9). Точно так же, как в случае данных других типов, между именем переменной и приписываемым ей объектом помещается знак равенства. Единственное отличие в том, что здесь оператор должен начинаться с ключевого слова Set.

Очистка объектной переменной

Когда доступ к объекту уже не нужен, неплохо освободить объект от привязки к переменной. В результате программа сможет использовать участок памяти, ранее занятый объектом, а вы будете уверены в том, что ваш программный код не внесет по ошибке ненужные изменения в данный объект.

Сделать это совсем просто. С помощью оператора Set присвойте переменной ключевое слово Nothing, как в следующем примере:

```
Set objPriceIsNoObject = Nothing
```

Создание новых объектов

Если объект, с которым вы собираетесь работать, еще не существует, вам придется его создать. В простых VBA-программах для этого используется метод Add, предназначенный для создания объектов, встроенных в ваше VBA-приложение, т.е. в приложение, с которым ассоциируется ваш проект. (В некоторых приложениях, возможно, именем такого метода будет что-то типа AddShape или AddDocument — проверьте в справочной системе приложения, чтобы знать точно.)

Использование метода Add

Самое главное — выяснить, метод Add какого именно из объектов следует применить. Цель — найти объект, который должен служить *контейнером* для создаваемого объекта. Обычно (но не всегда) подходящий контейнер представляет собой коллекцию.

Например, предположим, что нужно создать новый объект Layer в Visio. Объект Layer соответствует слою изображений в Visio. т.е. группе графических элементов, с которыми можно работать одновременно. Каждый объект Layer принадлежит объекту Layers, представляющему коллекцию из (одного или нескольких) слоев изображений. Исходя из этих соображений, для создания нового слоя вы должны выбрать метод Layers . Add.

Конечно, необходимо идентифицировать тот конкретный объект Layers, в котором VBA придется создать ваш новый объект Layer. Коллекция Layers принадлежит конкретному объекту Page в объекте Document. Поэтому полностью оператор для создания нового объекта Laye r должен выглядеть примерно так:

```
ThisDocument;. Pages (2} . Layers . Add ( "Новый слой")
```

Объект Layer тоже имеет метод Add, но его нельзя использовать для создания подобного нового слоя. Метод Add объекта Layer добавляет объект Layer в данный слой, а не создает еще один такой же Layer.

Точно так же, если нужно добавить новый слайд в презентации PowerPoint, понадобится примерно такой оператор:

```
ActivePresentation.Slides.Add 1, ppLayoutTextAndClipart
```

Как видите, новый слайд тоже добавляется в подходящий объект-контейнер— коллекция `Slides`.

Создание переменной для создаваемого объекта

Ввод длинных ссылок на объекты не приносит слишком много радости,— а именно такие ссылки содержат операторы, использующие метод Add, — поэтому имеет смысл при создании объекта сразу же создать и переменную для этого нового объекта. Тогда в следующий раз вместо длинной ссылки на объект в программном коле можно использовать имя переменной. Вот пример использования такого подхода:

```
Dim objМойМалыш As Glide
Set objМойМалыш = ActivePresentation.Slides.Add 1, _
    ppLayoutTextAndClipart
```

*Создание объектов с помощью **New** и CreateObject*

В иных случаях для создания объектов используются другие приемы. Вот несколько вариантов на выбор.

- ✓ Создание новой копии объекта, уже существующего в проекте.
- ✓ Использование объекта из *другого* приложения или компонента ActiveX (COM).
- ✓ Создание экземпляра объекта на основе класса, созданного вами в модуле класса.

В зависимости от ситуации, при этом используется либо ключевое слово New в объявлении переменной или операторе `Set`, либо функция `CreateObject`. Здесь я не собираюсь объяснять, как конкретно используется такая техника — она относится к тонким вопросам программирования и обсуждается в главах 10 и 19. Я только обращаю каше внимание на существование ключевого слова `New` и функции `CreateObject`, с помощью которых создаются обычные объекты.

Использование оператора Is

Раз уж вы собираетесь использовать переменные для хранения ссылок на объекты, наступит время, когда вы захотите выяснить, ссылается ли данная переменная на тот же объект, что и другая переменная или объектное выражение.

Проверить идентичность двух ссылок на объекты можно с помощью оператора `Is`. Значением соответствующего выражения будет `True`, если ссылки идентичны, и `False`, если ссылки указывают на разные объекты. Вот фрагмент программного кода, иллюстрирующий использование оператора `Is`:

```
Dim objObject1 As Object, objObject2 As Object
...
If objObject1Is objObject2 Then
    MsgBox "Это тот же самый объект!"
Else
    MsgBox "Это разные объекты."
End If
```

С помощью оператора Is можно сравнить объектную переменную и с объектным выражением, например, так:

```
If objObject3 Is ThisDocument.Pages(2).Shapes(3) Then
...
```

Обратите внимание, оператор Is (как и любой другой оператор) нельзя использовать для выяснения идентичности *содержимого* двух объектов.

Эффективная работа с объектами в программе

Для упрощения работы с объектами в программе в VBA можно использовать две многострочные программные конструкции, называемые операторами With и Each...Next.

Использование оператора With

И когда для ссылки на объект используется мнемоническая переменная с коротким именем, и когда используется вполне инструктивное объектное выражение, их приходится вводить снова и снова. Но, оказывается, даже этого можно избежать.

Если в программе один и тот же объект используется в нескольких идущих один за другим операторах, то оператор With позволяет указать объект только один раз. Это не только избавит вас от необходимости печатать имя объекта для каждого из операторов, но и делает программный код более понятным, а его выполнение более быстрым. Например:

```
With objПолнаяОбъективность
     .Name = "Опрос общественного мнения" ' установка свойства Name
     .DisplayName ' вызов метода DisplayName
     sngРегион = .Area ' выяснение значения свойства Area
     intПодтасовка = .Rotate(60) ' вызов метода Rotate (Поворот)
          ' и сохранение возвращаемого им значения в переменной
End With
```

Как видите, конструкция With...End With может включать операторы, в которых читаются и устанавливаются свойства, вызываются методы. Обратите внимание на то, что данная конструкция не задает цикл — входящие в нее операторы выполняются только один раз.

Конструкции, использующие With, можно вкладывать одну в другую. Это удобно, когда нужно выполнить ряд действий и по отношению к некоторому объекту, и по отношению к одному из содержащихся в нем объектов. Пример предоставляет следующий фрагмент программного кода. В нем выполняются самые различные действия с объектом Block в AutoCAD. Во вложенной структуре With обрабатывается один из графических объектов, содержащихся в объекте Block. Разобраться в деталях вам помогут комментарии.

```
With Block.Item("Монстр")
 ' Объект Block с именем "Монстр" делает следующее:
      ' переименовывает Block с помощью свойства Name
      .Name = "Душечка"
```

```
        ' выясняет в нем число объектов с помощью свойства
Count
        intЧислоОбъектов = .Count
        ' вызывает метод AddCircle со значениями
        ' для центра = 0 и для радиуса = 5
        AddCircle (0#, 5#)
        With .Item (1)
        ' эта вложенная структура With ссылается на первый
        ' графический объект в объекте Block
            ' графический объект перемещается с помощью
метода Move
            .Move (15, 20)
            ' изменяется свойство Color этого графического
объекта
            .Color = 221
        End With
End With
```

Данный пример не иллюстрирует, что вы можете использовать значения, возвращаемые свойствами или методами объекта, в условных операторах и других выражениях. Например, во внешнюю структуру With можно было бы добавить следующий фрагмент программного кода:

```
intНовыйЦвет = InputBox ("Введите число, задающее " _
& "цвет первого объекта в блоке " & .Name)
If .Count > 12000
    MsgBox "Да этот блок просто гигант!"
End If
```

Использование структуры For Each...Next

Вообще-то, *управляющим структурам* VBA (группам операторов, предназначенных для управления потоком выполнения программы) посвящена глава 8, но одну из таких структур мы рассмотрим здесь, потому что она используется только с объектами. Как аналог идеи цикла For..Loop, в VBA структура For Each...Next представляет множество операторов, относящихся ко всем объектам в некоторой коллекции. Правда, хотя обе структуры выглядят очень похоже, между ними есть существенные различия по сути, и For Each... Next проще в использовании. Вот синтаксис:

For Each *элемент* In *группа*
    (операторы для выполнения при каждом проходе цикла)
Next *элемент*

Ключевым различием в использовании For Each...Next и For...Next является то, что здесь вам не требуется указывать число повторений выполнения цикла — это за вас выяснит VBA. В операторе For Each с помощью переменной элемент определяется тип объекта в коллекции, а с помощью аргумента *группа* задается коллекция, с которой нужно работать.

Структуры For Each...Next в действии

Следующий простой пример работает в AutoCAD, где коллекция Blocks является коллекцией объектов Block и каждый объект Block содержит неограниченное число графических объектов типа Cone (конус) или 3DFace (трехмерная поверхность). В этом фрагменте программного кода всего лишь отображается имя каждого объекта Block из коллекции Blocks:

```
Dim objB As Block
For Each objB In Blocks
    Debug.Print objB.Name
Next objB
```

Ясно, что этот пример мир не переворачивает, но вы должны видеть его потенциальную мощь. Структура For Each. . .Next может вызывать любую комбинацию действий, доступных после вызова методов и свойств объектов, входящих в коллекцию.

Принудительный выход из структуры For Each...Next

Когда вам нужна определенная игла в коробке с иглами, другие иглы вас не интересуют. Чтобы выполнить определенные действия по отношению к одному конкретному объекту в коллекции, используйте внутри структуры For Each. . .Next оператор If. . .Then. Когда условие If. . .Then обнаружит объект вашей мечты, дальше искать станет бессмысленно, и тогда быстрый выход из цикла обеспечит оператор Exit For.

Снова обратившись к коллекции Blocks из AutoCAD, рассмотрим следующий пример:

```
Dim objK As Block
For Each objK In Blocks
    If objK.Name = "Ерунда"
        objK.Delete
        Exit For
    End If
Next objK
```

В этом примере вложенная конструкция If. . . Then используется для того, чтобы обнаружить объект Block с именем "Ерунда" среди всех объектов Block в коллекции Blocks. После того — и только после того — как нужный Block найден, выполняются операторы внутри структуры If. . .Then. "Выполняются" здесь относится, главным образом, к первому из операторов, который удаляет объект Block. Миссия завершена. Цикл больше не нужен, поэтому Exit For прерывает его, и программа может продолжить свою работу.

Глава 13

Тонкости хранения данных: массивы и коллекции

В этой главе...

- Использование массивов для управления наборами элементов одного и того же типа
- Многомерные массивы
- Объект Collection как альтернатива массивам
- Создание своих собственных типов данных для работы с информацией, состоящей из родственных данных различных типов

К этому моменту вы должны чувствовать себя уверенно при работе с переменными, хранящими данные любых типов,— от обычных чисел и строк текста до сложных, имеющих мощный потенциал, объектов. Но бывает, что ни один из встроенных типов данных VBA разработчика не устраивает. Часто необходимо работать с некоторым набором однотипных данных как с единым целым. В таких случаях используются *массивы* — структурированные боксы для хранения множеств элементов данных одинакового типа.

Работать с множествами элементов различных типов позволяют *объекты* Collection (коллекции), которые создаются на основе соответствующего встроенного класса VBA, При некоторых ограничениях использование коллекций бывает целесообразнее, чем применение массивов.

Иногда удобнее создать свой собственный *пользовательский тип данных.* Как и переменные, пользовательские типы данных упрощают восприятие программного кода и позволяют обращаться к отдельным, но родственным в некотором смысле, элементам как к частям единого целого. Но массив данных пользовательского типа оказывается еще удобнее, если приходится иметь дело с группами сложных составных наборов информации.

Знакомство с массивами

Предположим, у вас есть набор чисел, представляющих цены, оценки за экзамен, расстояния от Земли до каких-то астрономических объектов или нечто другое. Представьте себе, что вы печатаете этот набор чисел в виде списка на листе бумаги, каждое число — в отдельной строке. То, что у вас получится, можно назвать простым массивом. Например:

Гарантированно выигрышные лотерейные номера

```
214236
545273
371453
891982
000000
941241
```

В списках, подобных этому, отдельные элементы имеют вполне определенные значения, но не имеют специальных идентификационных пометок. Если нужно, чтобы на один из элементов списка обратил внимание кто-то другой, вам придется сказать приблизительно следующее: "Это третий элемент в списке лотерейных номеров". Примерно так же VBA работает с массивами.

Ссылка на элемент массива

Каждый массив в VBA имеет имя, т.е. аналог заголовка списка на листе бумаги. Чтобы работать с отдельным элементом массива, нужно сослаться на него по имени массива и *индексу* — целому числу, соответствующему месту элемента в массиве. Например, выражение `intLottoArray(3)` ссылается на третий (или четвертый, в зависимости от системы нумерации) элемент массива с именем `intLottoArray`. Как вы, наверное, догадались, `int` в начале имени массива говорит о том, что в этом массиве предполагается хранить целые значения. Поэтому можно утверждать, что данные, хранящиеся в `intLottoArray(3)`, представляют собой целое число.

Данные массива

При работе с массивами нужно помнить следующее.

- Можно создавать массивы данных любых типов. VBA с успехом хранит в массивах строки, даты, денежные значения и данные любых числовых типов.
- В одном массиве могут храниться данные только одного типа. Нельзя создать массив с раздельными ячейками для хранения и данных типа `Date`, и данных типа `String`.

Правда, второе ограничение очень легко преодолеть. Как вы уже знаете, тип данных `Variant` допускает хранение данных любого из допустимых в VBA типов, и массивы данных типа `Variant` тоже вполне допустимы. Здесь следует иметь в виду, что информация, хранимая в виде данных типа `Variant`, занимает существенно больше места в памяти, чем та же информация в виде данных более определенного типа; для массивов произвольной длины это может потребовать дополнительно очень значительного расхода памяти. Но необходимость поместить в один массив неопределенной длины набор данных различных типов возникает крайне редко.

С другой стороны, довольно часто необходимо работать с наборами родственных в некотором смысле элементов разного типа. Именно такие наборы составляют структуру типичной базы данных. В базе данных для хранения адресов, например, каждая запись представляет собой некоторый набор элементов информации (имя и адрес), относящихся к одному объекту (человеку). Для подобных данных как раз и предлагается создавать пользовательские типы данных — как это сделать, вы узнаете чуть позже, в разделе "Определение своих собственных типов данных").

Использование многомерных массивов

Прежде чем заняться индивидуальными значениями, хранящимися в массиве, рассмотрим несколько общих вопросов, относящихся к массивам. Прежде всего следует сказать, что массивы могут быть *многомерными*. Простой список лотерейных номеров из примера в начале этой главы является одномерным массивом. Моделью двумерного массива может служить нижеприведенная табличка данных, организованных в строки и столбцы.

Ежедневные наблюдения за курами тремя анонимными биологами

12	2	21
4	9	3
11	O	O

Представить трехмерный массив тоже не слишком сложно. На рис. 13.1 показан трехмерный массив в виде диаграммы.

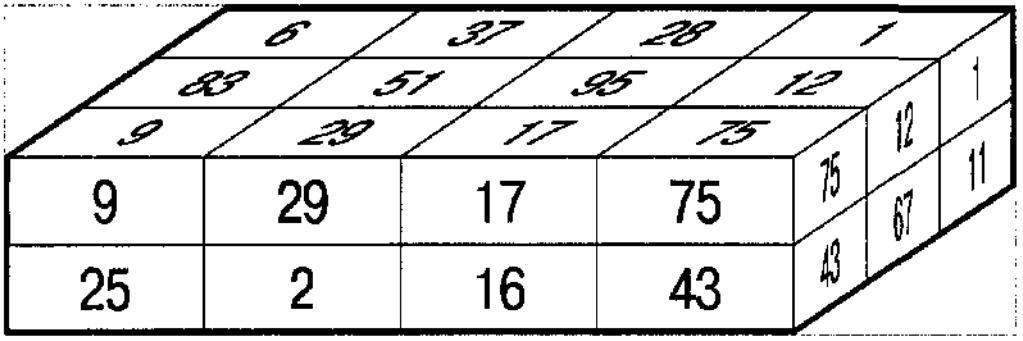


Рис. 13.1. Эта диаграмма представляет трехмерный массив в виде набора кубиков с числами

А вот представить себе массив большей размерности уже не так просто, хотя для VBA это не составляет никаких проблем — массивы VBA могу иметь размерность вплоть до 60.

Объявление массивов

Как и обычные переменные, массивы должны быть объявленными. При этом объявление массива по форме отличается от объявления переменной только в одной дополнительной детали. Вот несколько примеров:

```
' Объявление одномерного массива данных типа Date:
Dim datTimeOfImpact (cdatMaxObservations) As Date
' Объявление массива типа Currency без объявления размера:
Public curPriceQuotes () As Currency
' Объявление четырехмерного массива типа Integer:
Dim intArrayOfIntegers (34, 13, 29, 4) As Integer
```

Как видите, разница объявлений для обычных переменных и массивов заключается в том, что в объявлении массива за именем следуют скобки, в которых может не содержаться ничего, а могут содержаться значения, задающие размеры массива по каждому из его измерений. (Обратите внимание, что в первом из вышеприведенных примеров размер одномерного массива задается именованной константой.)

Не забудьте при объявлении массива указать тип данных, включив в объявление массива As `Type` с подходящим ключевым словом, задающим тип. Если не указать тип, VBA создаст массив типа `Variant`, в результате чего требования вашей программы к памяти возрастут, а ее выполнение замедлится. Массивы данных типа `Variant` допустимы и в некоторых ситуациях могут быть вполне подходящим решением, но всегда, когда это возможно, лучше назначать массивам конкретный тип данных.

Общее число элементов данных в массиве равно произведению величин, характеризующих каждое из его измерений, — вы об этом уже знаете. Но здесь нужно обратить ваше внимание на одну деталь. Число элементов в массиве `intArrayOfIntegers` в третьем объявлении из предложенного выше примера равно 35x14x30x5, так что это маленькое объявленьице резервирует место для масснвчика из 73500 элементов.

Вы можете спросить, почему я считаю, что размер массива по каждому из его измерений на единицу больше соответствующих значений в объявлении (35 вместо 34 и т.д.). Ответ вы найдете ниже.

Нумерацияэлементовмассива

Если вы не укажете иное, элементы массива индексируются (т.е. нумеруются) начиная с 0; говоря иначе, первым в массиве будет элемент с индексом 0. По этой причине значение, задающее размерность массива в объявлении, должно быть на единицу меньше числа элементов, которые предполагается иметь. Например, если предполагается создать массив из 10 элемснтов, для размера нужно указать значение 9.

Об этом не следует забывать, когда придется обращаться к элементам массива. Ссылка типа `intМассив(1)` на самом деле означает обращение ко второму элементу в массиве.

Если же вам не нравится отсчет от 0, можно начать нумерацию элементов в любом массиве от другого числа — обычно от 1. Чтобы заставить VBA нумеровать все массивы от 1, поместите в раздел объявлений модуля (перед всеми процедурами) оператор

```
Option Base 1
```

Действие этого оператора касается только массивов внутри соответствующего модуля, поэтому при необходимости нужно будет разместить подобные операторы внутри каждого модуля.

Можно задать схемы нумерации и для каждого из измерений в отдельности. Например;

```
Dim sngNumbers (55 To 75, 7 To 16, 99)
```

В этом примере объявляется трехмерный массив, в котором по первому измерению имеется 21 место, с началом нумерации от 55 до 75; по второму — 10 мест с началом нумерации от 7, а последнее измерение предполагает наличие 100 мест с нумерацией от 0.

Объявление массивов фиксированныхразмеров

Если при объявлении массива указать значение для его размера, то размер массива останется фиксированным в ходе всего выполнения программы — программа не сможет сделать массив меньше или больше. Чтобы объявить массив фиксированного размера, укажите в скобках в его объявлении конкретные значения для каждого из измерений массива. Например, оператор

```
Edra strХорошиеМысли (9, 19) As String
```

объявляет двумерный массив, который всегда будет содержать 10 строк и 20 столбцов строковых данных (конечно, массив на самом деле не содержит данные в строках и столбцах, *но* мысленное представление массива в виде подобной таблицы очень удобно).

Фиксирование размера массива при его объявлении определенно можно рекомендовать, если точно известно, что изменение размера массива не потребуется. В этом случае вы застрахованы и от непредусмотренных изменений размера массива в программе. Правда, потенциальным недостатком фиксированного размера является то, что вы не сможете освободить *в* программе занимаем>ю массивом память для других нужд.

Объявлениединамическихмассивов

Объявляйте массив как *динамический* в следующих случаях.

- ✓ Если вы не знаете и не можете узнать размер массива ло выполнения программы.
- ✓ Если знаете, что размер массива в ходе выполнения программы будет меняться.
- ✓ Если после завершения использования массива хотите освободить занимаемую им память для других целей (большие массивы могут занимать немало памяти, которую можно освободить для использования динамических массивов).

Чтобы объявить динамический массив, просто не указывайте размер массива при его объявлении. Например, оператор

```
Dim dateДниРождения () As Date
```

подготовит VBA к использованию массива элементов типа Date, на самом деле не создавая пока массив.

Переопределение динамических массивов

Динамический массив не может хранить данные до тех пор, пока вы не создадите массив реально, указав его размер. Для этого используется оператор ReDim, как в следующем примере с одномерным массивом:

```
ReDim dateДниРождений (intЧислоДнейРождения - 1)
```

Обратите внимание на то, что размер создаваемого этим оператором массива задается выражением, использующим переменную, в предположении, что значение этой переменной уже определено в программе. Значение переменной приходится уменьшать на 1, чтобы привести число элементов массива в соответствие с предлагаемой VBA нумерацией массивов от 0, если, конечно, вы не изменили систему нумерации, принятую по умолчанию, как описывалось в разделе "Нумерация элементов массива".

Оператор ReDim должен использоваться внутри процедур. Один и тот же массив можно переопределять столько раз, сколько нужно, задавая совершенно новые числа размерности и размеров. Только не забывайте, что при обычном переопределении массива его содержимое полностью уничтожается. Нельзя также изменить тип данных, назначенных массиву, разве что при определенных условиях, объяснять которые здесь мне кажется излишне сложным.

Сохранение данных при переопределении массива

Чтобы при переопределении массива сохранить часть данных, используйте вместе с оператором ReDim ключевое слово Preserve. Такая последовательность операторов объявляет динамический массив, создает его как двумерный массив заданного размера, а затем увеличивает размеры массива по второму измерению, не разрушая уже имеющиеся данные:

```
Dim dblGalacticMasses () As Double
...
ReDim Preserve dblGalacticMasses (1 To 30, I To 50)
...
ReDim Preserve dblGalacticMasses (1 To 30, 1 To 100)
```

На самом деле возможности Preserve весьма ограничены — при использовании этого ключевого слова не допускается менять размерность массива и можно менять размер только последнего измерения (но все равно необходимо в операторе ReDim Preserve указывать параметры остальных измерений). При этом можно сделать размер массива меньше, но данные, хранившиеся в удаленных элементах, будут утеряны навсегда.

Обращение к элементам массива

Чтобы в программе использовать конкретный элемент массива, напечатайте имя массива, за которым следуют скобки с указанным в них индексом этого элемента. *Индекс* должен задавать целые значения для каждого из измерений массива. Например, выражение `strSayings(4,6)`, очевидно, однозначно идентифицирует строковые данные из строки 4 и столбца 6 в двумерном массиве строк.

Не забывайте, что числа или значения, которые вы должны указать в индексе для идентификации элемента массива, зависят от системы нумерации, указанной вами при объявлении массива. Если массив объявлен оператором `Dim strSayings(10,20) As String`, то выражение `strSayings(4,6)` идентифицирует данные из пятой строки седьмого столбца массива. Но если массив объявлен `Dim strSayings(4 To 10, 6 To 50) As String`, то же самое выражение означает данные из первой строки первого столбца.

Использование элементов массива в программе

В рамках описанной системы можно использовать элементы массива как обычные переменные. Вы можете сделать следующее:

- ✓ присвоить элементу массива значение; в следующем примере значение присваивается конкретному элементу трехмерного массива данных типа `Currency`:

```
curBigDough(5,8,19) = 27.99
```

- ✓ присвоить значение, хранящееся в массиве, некоторой переменной, например:

```
datThatDate = datTheseDates(25,10)
```

- ✓ использовать значение элемента массива в выражении, например:

```
intA = 35 * (intB + intCounts(3,2))
```

Использование переменных в индексах

Конечно, в индексах можно использовать не только буквальные значения. Задание индекса с помощью переменных позволяет выбрать в программе тот элемент массива, который нужен в данный момент. Это демонстрирует последняя строка следующего фрагмента программы:


```
Sub FortuneTeller()
Dim strTodaysFortune(1 To 10)
Dim intUserChoice As Integer
strTodaysFortune(1) = _
 "Вы станете богатым и знаменитым на 15 минут."
strTodaysFortune(2) = _
"Вам предстоит обед с высоким незнакомцем."
strTodaysFortune(3) = "Стоимость ваших акций удвоится!"
strTodaysFortune(4) = _
"Вы вспомните, где положили свои ключи."
strTodaysFortune(5) = _
"Вы получите огромный букет от анонимного почитателя."
strTodaysFortune(6) = "Вы опоздаете на обед."
strTodaysFortune(7) = "Все ваши мечты сбудутся."
strTodaysFortune(8) = _
"Вам вернут взятые у вас библиотечные книги."
```

```
strTodaysFortune(9) = _
"Никто не заметит ваших разных носков."
strTodaysFortune(10) = _
"Вы встретите своего старого друга в супермаркете."
intUserChoice = InputBox("Чтобы узнать свое будущее," _
     & " введите число от 1 до 10")
MsgBox (strTodaysFortune(intUserChoice))
```

Кстати, этот программный код можно существенно улучшить, если добавить в него обработку неправильного ввода, возможного при ответе пользователя на запрос функции InputBox. О проблемах проверки данных при вводе говорилось в главе 10.

Выяснение размера массива

Чтобы выяснить, как много элементов может вместить некоторое измерение массива, используйте функцию Ubound, которая особенно полезна при работе с динамическими массивами, поскольку в разные моменты выполнения программы динамический массив может содержать разное число элементов. Синтаксис функции UBound следующий:

```
UBound(имя_массива,измерение)
```

Аргумент имя_массива, очевидно, задает имя массива, к которому нужно обратиться, а аргумент измерение задает целое число, которое подскажет VBA, какое измерение массива вас интересует. Если измерение не указано, функция UBound возвратит размер первого измерения массива.

Заполнение массива данными... навалом

При первом создании массива его элементы не содержат никакой действительной информации. В некоторый момент в программе вам необходимо будет заполнить вакантные места массива подходящими данными. Если нужно разместить сразу много данных, лучше всего для этого подойдут вложенные циклы For. . . Next, по одному на каждое измерение массива. Для счетчика цикла обычно выбирается временная переменная, объявляемая в процедуре, содержащей все эти циклы.

Этот подход иллюстрируется следующей небольшой программой, в которой трехмерный массив заполняется последовательными целыми значениями начиная с 1. Вот ее программный код:


```
Const Size As Integer = 3
Dim dblMatrix (1 To Size,1 To Size,1 To Size) As Double
Sub ArrayFiller()
Dim I As Integer,J As Integer,K As Integer,X As Integer
X = 1
For I = 1 To Size
     For J = 1 To Size
          For K = 1 To Size
               dblMatrix(I, J, K) = X
               X = X + 1
          Next K
     Next J
Next I
End Sub
```

Порядок вложения циклов будет определять порядок заполнения массива дачными. В свою очередь от порядка, в котором заполняются элементы массива, часто зависят и их значения.

Объяснять это только на словах слишком сложно, поэтому я собираюсь использовать рисунки. Снова обратимся к предыдущему примеру программного кода. Представим трехмерный массив в виде куба, сложенного из маленьких кубиков, означающих элементы массива. Иллюстрация этого мысленного эксперимента показана на рис. 13.2.

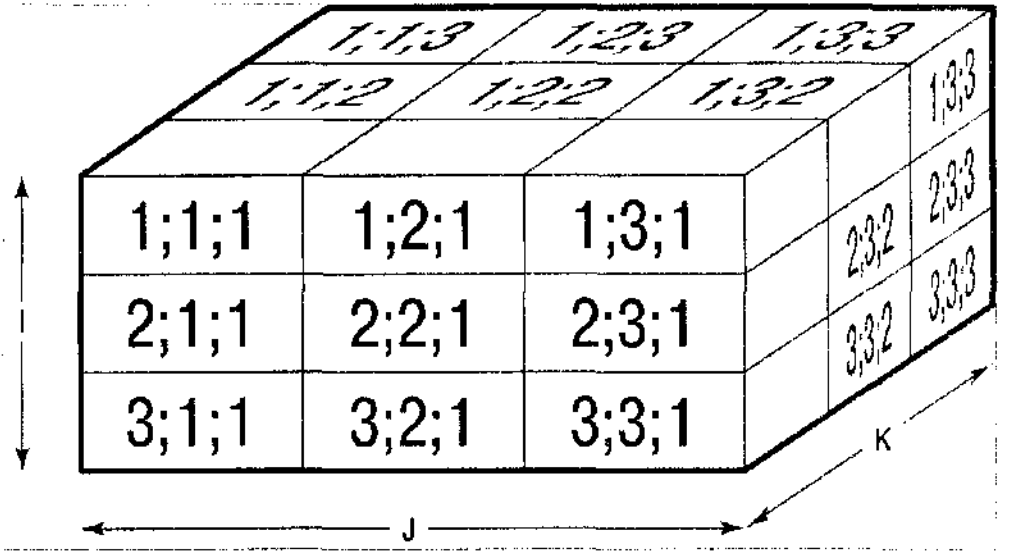


Рис. 13.2. Этот чертеж представляет массив для хранения данных; стрелки с буквами соответствуют измерениям массива

Обратите внимание на то, что в главной строке этого программного кода, dblMatrix(I, J, K) = X, переменные I, J и K представляют соответственно первое, второе и третье измерения массива. На рис. 13.2 я произвольным образом назначил эти буквы сторонам куба, задающим его измерения.

Теперь посмотрите, как эти переменные используются в циклах For... Next. Счетчиком внешнего цикла является I, следующего — J и последнего, внутреннего,— K. Нужно понять, что VBA начнет выполнение с внутреннего цикла, пройдя к нему сквозь все содержащие его циклы. Кубы на рис. 13.3 показывают два последовательных момента в самом начале заполнения массива из этого примера.

Л пока этот внутренний цикл выполняется, VBA не меняет значения переменных I и J. Когда внутренний цикл первый раз пройден полностью, первый раз получит возможность сделать шаг следующий цикл — цикл J. Это увеличит значение J — и снова начнется выполнение внутреннего цикла от начала до конца. Значение I изменится только тогда, когда цикл J выполнен все три раза. После увеличения I начинает заполняться следующий слой массива (рис. 13.4).

Надеюсь, теперь вы поймете, как отражается изменение порядка вложения циклов на порядке заполнения массива. Предположим, что порядок вложения циклов изменен на обратный, и счетчиком внешнего цикла теперь стало K. Вот как должен выглядеть при этом соответствующий программный код (обратите внимание, что в операторе присваивания переменные остались на прежних местах):

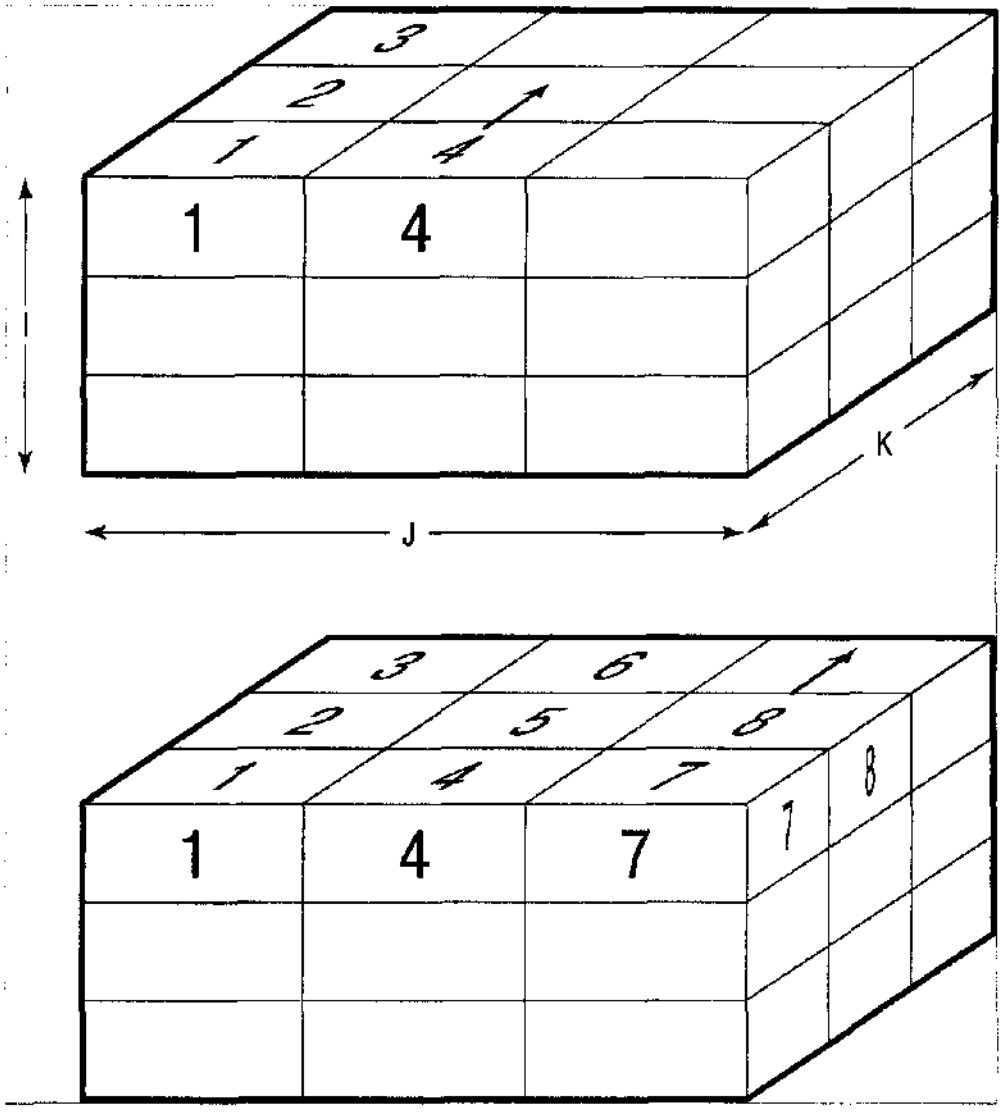


Рис. 13.3. Эти два чертежа иллюстрируют последовательность, в которой процедура-пример запол массив по мере выполнения внутреннего цикла `For..Next`

```
For K = 1 To Size
     For J = 1 To Size
          For I = 1 To Size
               dblMatrix(I, J, K) = X
               X = X + 1
          Next I
     Next J
Next K
```

В результате массив будет выглядеть так, как показано на рис. 13.5.

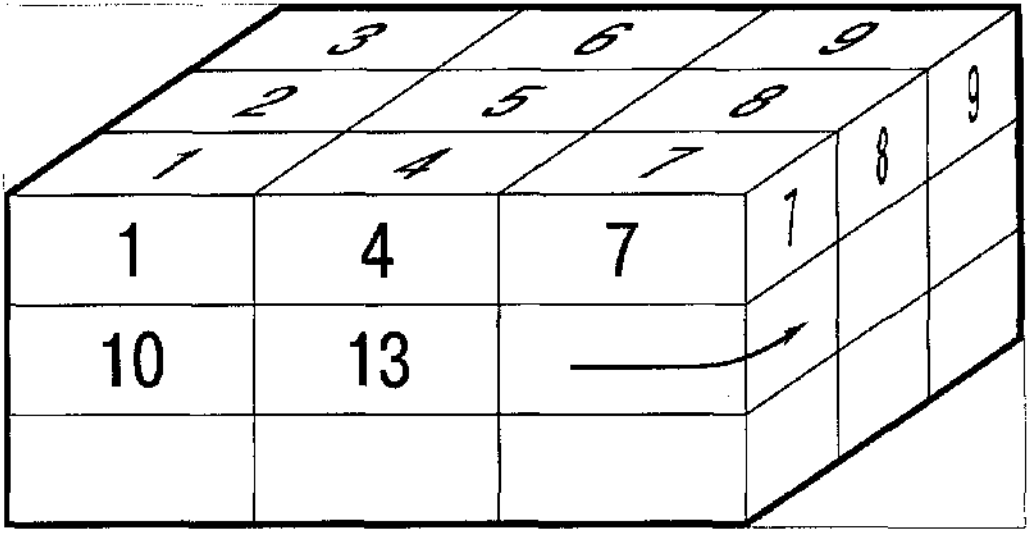


Рис. 13.4. Второй слой массива начнет заполняться только после того, как изменит значен внешнего цикла For..Next

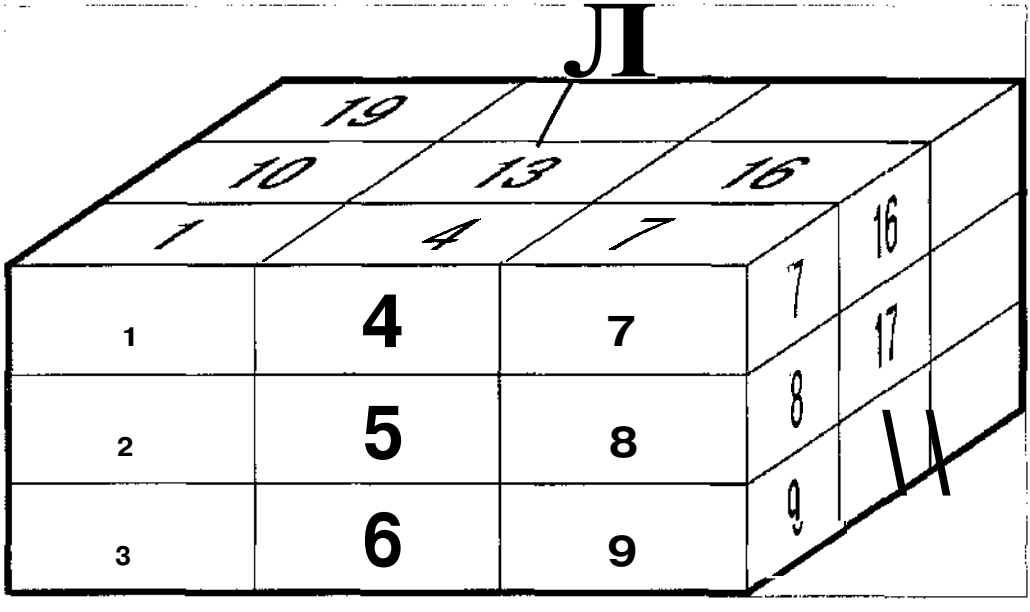


Рис. 13.5. Этот чертеж иллюстрирует порядок заполнения массива измененной процедурой

Массив будет содержать те же значения, что и в первом случае, но они хранятся иными элементами массива.

Копирование одного массива в другой

В VBA 6 можно присвоить содержимое одного массива другому с помощью простого оператора наподобие

```
strМассивКопия = strИсходныйМассив
```

Массив слева (тот, которому присваиваются значения) должен быть динамическим, и VBA переопределит его автоматически в соответствии с размерностью и размерами массива справа. Получающий значения массив должен также допускать хранение данных того типа, который назначен данными исходного массива.

В VBA 5 вам придется копировать данные массива поэлементно, примерно так:

```
' выяснение размера исходного массива и
' соответствующее переопределение второго массива
intРазмерМассива = UBound(strИсходныйМассив)
ReDim strМассивКопия(intРазмерМассива)


' выполнение копирования
For I = 0 To intРазмерМассива
    strМассивКопия(I) = strИсходныйМассив(I)
Next I
```

Управление наборами данных с помощью объектов Collection

Если нужно работать с наборами элементов информации, создайте для этой информации объект Collection (Коллекция). Как уже говорилось в главе 12, в VBA родовой класс Collection предназначен для хранения практически всего, что только возможно. На основе этого класса можно создать любое число объектов Collection. Коллекции используются для хранения других объектов, но точно так же коллекции можно использовать и для хранения данных почти любых из допустимых в VBA типов.

Коллекция может содержать только простой список элементов, похожий на одномерный массив. Но поскольку коллекции относятся к простейшим из массивов, за счет специализации вы получаете выигрыш в качестве обслуживания.

Оценка преимуществ использования коллекций

Если вы чувствуете себя вполне уверенно при работе с объектами VBA, то использование объектов Collection для управления множествами элементов может оказаться для вас более простым, чем использование массивов. Методы Add и Remove позволяют без труда изменять размеры коллекции, и при этом вероятность возникновения ошибок будет меньше, чем при использовании оператора ReDim в разных частях программы. (Эти преимущества становятся особенно важными, если необходимо часто добавлять и удалять элементы группы.)

Кроме того, элементам коллекции можно назначить имена, по которым позже получают доступ к элементам. При этом вы не только получите возможность выбрать описательные имена, которые легко запомнить, но ускорите доступ к данным, особенно при числе элементов, превышающем 100.

Одним из отрицательных побочных эффектов при использовании коллекций является увеличение расхода памяти — коллекции для хранения информации используют тип Variant, а данные типа Variant занимают больше памяти, чем данные других типов. В больших коллекциях такого рода перерасход памяти весьма значителен.

Но самое главное ограничение при использовании объектов Collection — запрет па хранение в них данных пользовательского типа, т.е. их нельзя непосредственно использовать с базами данных (см. ниже раздел "Определение своих собственных типов данных"). Правда, с помощью несложных приемов программирования это ограничение довольно просто обойти. И кроме того, серьезно работая с VBA с базами данных, вы все равно, скорее всего, будете использовать готовые объекты из ядра Microsoft Jet (DAO) и библиотек ODBC, а не объекты, созданные кустарным способом.

Создание объектов Collection

Объекты Collectionсоздаются в программе точно так же, как и любые другие объекты, с использованием ключевого слова As для определения типа. Как и с другими объектами, у вас есть на выбор два варианта.

- ✓ Можно объявить имя переменной для объекта и затем использовать оператор Set, чтобы создать ее. При этом в операторе Set нужно использовать ключевое слово New, чтобы создать новую коллекцию. Например:

```
Dim colMixedBag As Collection
...
Set colMixedBag = New Collection
' создание коллекции
colMixedBag.Add "Howard, Ethel"
' добавление элемента
```

- ✓ Можно обязать VBA создать объект автоматически при первом использовании переменной в программном коде. Для этого нужно поместить ключевое слово New в объявление переменной;

```
Dim colSetOfStuff As New Collection
...
' следующий оператор создает коллекцию
' при добавлении в нее целого значения
colSetOfStuff.Add intStuffing
```

Добавление данных в коллекцию

Для заполнения созданной коллекции данными используйте метод Add (Добавить) точно так же, как при добавлении объектов во встроенные коллекции VBA-приложения (см, главу 12). Примеры использования метода Add вы найдете и в предыдущем разделе.

Синтаксис метода Add выглядит так:

Add (*элемент*[, *имя*][, before *индекс*][, after *индекс*])

При этом выражение *элемент* обязательное, оно может быть буквальным значением, переменной, ссылкой на объект или более сложным выражением, составленным из этих компонентов, — в общем, всем, что возвращает значение, распознаваемое VBA. Остальные аргументы метода Add необязательны, они обсуждаются в следующих разделах.

Присваивание имен элементам

В общем-то, всегда можно обратиться к элементу коллекции по его индексу в этой коллекции, но зачастую удобнее назначить элементу информативное имя. Для этого нужно при добавлении элемента в коллекцию указать его имя в виде строки:

```
colFinancials.Add 14323.44, "Продажи за февраль"
```

В этом операторе в коллекции `colFinancials` добавляется значение `14323.44`. Одновременно для этого элемента создается имя. Вполне допустимо при задании имени использовать переменную типа `String`.

Имя не только легче запомнить, чем индекс, оно, к тому же, обеспечивает *единственный* надежный способ доступа к конкретному элементу коллекции. При использовании методов Add и Remove позиции элементов данных в коллекции могут меняться. Даже если элемент с №69 станет 29-м, вы все равно сможете воспользоваться значением этого элемента, если знаете его *имя*.

Имя элементу можно присвоить только с помощью метода Add. Чтобы присвоить имя элементу, уже существующему без имени, придется удалить имеющуюся копию с помощью `Remove` и использовать `Add` для добавления элемента снова.

Добавление элемента в заданном месте

Иногда удобно разместить элементы коллекции в определенном порядке. И даже если при этом нельзя добавлять элементы в нужном порядке последовательно, то все равно нет проблем — метод Add позволяет вставлять элементы туда, куда требуется, просто доя этого нужно использовать аргументы `before` (перед) и `after` (после).

Предположим, что вам нужно вставить новый элемент перед 35-м элементом в коллекции. Вы можете использовать для этого оператор следующего вида:

```
colAnimals.Add strSpecies, before 35
```

Добавленный элемент станет 35-м элементом коллекции, подвинув все последующие, (В отличие от массивов, элементы коллекций нумеруются с I.) Чтобы сообщить VBA, где вставить новый элемент, можно использовать и имя существующего элемента, например:

```
colОвощи.Add = strСорт, after "Помидор"
```

В данном случае VBA найдет в коллекции существующий элемент с именем Помидор и сразу после него вставит новый.

Ясно, что в оператор, использующий метод Add, можно включить либо `before`, либо `after`, но никак не оба этих аргумента.

Добавление нескольких элементов сразу

Для добавления в коллекцию множества элементов можно использовать цикл `For...Next`, подобно тому, как это делалось с массивами. Например:

```
Dim X As Integer, Y As Integer
Y = 12
For X = I To 30
     colHouseOfValues.Add Y * X
Next X
```

Удаление элементов

Говоря очевидное, сообщаю вам, что метод Remove удаляет элемент из коллекции. Объект для удаления можно задать либо с помощью индекса, либо с помощью имени, например:

```
colМинералы.Remove 2123
colМинералы.Remove "Боксит"
```

Не забывайте, что при удалении элемента, VBA, так сказать, "заполняет дыры" — номера всех элементов, следующих за удаленным, уменьшатся на 1.

Подсчеты в коллекциях

Запутаться в размерах коллекции несложно, особенно после многократного использования Add и Remove. Родовой объект Collection имеет только одно свойство — Count (подсчет), — но, в силу вышесказанного, это свойство оказывается жизненно важным. Значение этого свойства можно присвоить переменной, например, так:

```
intРазмерКоллекции = col20Вопросов.Count
```

Его можно также использовать в условном выражении, как здесь:

```
If colPrices.Count > 1000
     MsgBox "Слишком много позиций!"
End If
```

Кроме того, значение свойства Count можно использовать в цикле For . . . Next, чтобы выполнить некоторые действия по отношению ко всем элементам коллекции, например:

```
Dim Z As Integer
For Z = 1 To colPrices.Count
     MsgBox "Цена " & colPrices(Z)
Next Z
```

Но лучше для этого использовать цикл For Each. . . Next (см. главу 8).

Доступ к элементам коллекций

Чтобы идентифицировать конкретный элемент в коллекции, используется индекс элемента или его имя (если оно определено). Например, значение элемента коллекции можно присвоить переменной:

```
datДеньРождения = colДниРождения("Василий Али-Бабаевич")
```

К сожалению, вам придется отслеживать имена самим — VBA не предлагает возможности прочитать имена элементов коллекции. Кроме того, нельзя изменить имя элемента иначе, как удалив этот элемент из коллекции и затем добавив его вновь. Точно так же нельзя в существующий элемент коллекции поместить новое значение. Следующий оператор присваивания работать *не будет:*

```
colInventory(1465) = 119
```

Опять же, единственный способ изменения значения в коллекции — удаление соответствующего элемента и добавление нового с новым значением. Подобных ограничений нет для объектов Dictionary (словарь), которые обсуждаются в главе 14.

Использование коллекций с базами данных

Как уже было сказано, применение класса Collection VBA ограничивается тем, что в нем нельзя хранить данные пользовательских типов. С этим ограничением можно смириться, если выбрать подходящую систему создания имен элементов. В следующем примере при каждом проходе цикла For...Next в коллекции создается запись базы данных. Эта запись состоит из трех полей, каждое из которых идентифицируется именем, составленным из простого описательного признака и номера записи. Вот соответствующий программный код:

```
Dim I As Integer, strName As String
Dim strPhone As String, strAddress As String
For I = 1 To Total ' I идентифицирует номер записи
    ' получение данных для этой записи
    strName = InputBox("Введите имя для этой записи")
    strPhone = InputBox("Введите номер телефона для _ этой
записи")
    strAddress = InputBox("Введите адрес для этой записи")
    ' Добавление "полей" для этой записи в коллекцийе
    colDatabase.Add strName, "Имя" & I
    colDatabase.Add strPhone, "Номер телефона" & I
    colDatabase.Add strAddress, "Адрес" & I
Next I
```

Впоследствии можно извлечь информацию, хранимую в отдельной записи по описательному признаку и номеру, как в следующем операторе:

```
MsgBox colDatabase("Имя2")
```

Если вы собираетесь создавать реальные программы для управления информацией в базах данных, не используйте для этого коллекции объектов — они имеют слишком много ограничений. Вместо этого используйте подключаемые компоненты доступа к базам данных (соответствующие примеры вы найдете в главе 14). Если же вы хотите выполнить всю работу самостоятельно, создайте свои собственные классы для объектов, позволяющих работать с пользовательскими типами данных. Ваши объекты должны позволять хранить в них любые комбинации данных, которые могли бы извлекаться как свойства. Кроме того, неплохо было бы добавить программный код для проверки правильности сохраняемой в отдельных элементах информации. (Модули классов рассматриваются, хотя и не слишком подробно, в главе 14.)

Определение своих собственных типов данных

Если вы не математик, многомерные массивы данных одного типа вам, скорее всего, вряд ли понадобятся. В практических задачах управления данными чаще приходится работать с наборами элементов совершенно *разных* типов. В VBA-программе для управления такой информацией лучше всего создать *пользовательский тип данных.*

Обычно приводят набившую оскомину аналогию между такими структурированными данными и карточками каталога (рис, 13.6}, хотя персональные компьютеры уже давным-давно заполонили все вокруг, и, как мне кажется, сегодня вряд ли кто-то пользуется каталогами из карточек.

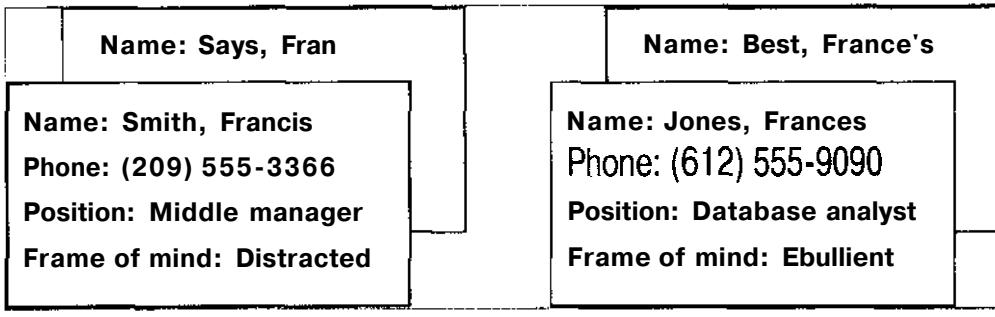


Рис. 13.6. На этих карточках хранится структурированная информация наподобие той, представлена данными пользовательских типов в VBA

Огромное количество баз данных фактически представляет такого же типа организованную структуру, но в электронном виде. *База данных* — это набор записей, каждая из которых состоит из полей для хранения отдельных элементов информации. Разные поля могут хранить данные совершенно различных типов (строки, числа, даты или что-то другое). Содержимое одного и того же поля при переходе от записи к записи может меняться, но тип хранимых в поле данных остается во всех записях одним и тем же.

Знакомство с пользовательскими типами данных

Пользовательский тип данных VBA представляет собой, так сказать, поселившуюся под одной крышей компанию выбранных вами типов данных. Определив пользовательский тип данных, вы получаете возможность объявлять переменные этого типа точно так же, как переменные встроенных типов.

Переменную пользовательского типа можно сравнить с отдельной карточкой в каталоге или одной записью в базе данных. А чтобы представить целый каталог с карточками или базу данных с записями, объявите массив данных пользовательского типа.

Объявление пользовательского типа данных

Для объявления пользовательского типа данных используется оператор `Type`, как в следующем типичном примере:

```
Type Персона
     intНомер As Integer ' номер работника
     strФамилия As String
     strИмя As String
     strАдрес As String
     lngТелефон As Long
     datДатаНайма As Date
End Type
```

Здесь объявляется новый пользовательский тип данных `Персона`. Как ясно показывают строки с отступами, этот тип состоит из целого значения, трех строковых значений, длинного целого и даты. (Обратите внимание на то, что при объявлении типа необходимо назначить

каждому элементу имя — простого указания типа данных недостаточно.) Когда вы объявите переменную типа Персона, в этой переменной автоматически будет предусмотрено место для всех шести элементов, перечисленных в объявлении типа.

Обратите внимание и на то, что для оператора Type требуется оператор End Type, завершающий блок объявления. Если вы забудете об этой завершающей строке, компьютер будет ругать вас.

И еще: пользовательский тип можно объявить только на уровне модуля (в разделе объявлений в самом начале модуля). Внутри процедуры объявить пользовательский тип нельзя.

Объявление переменных пользовательского типа

Как и встроенные типы данных, пользовательский тип остается абстрактным понятием до тех пор, пока вы не объявите переменную соответствующего типа. Для этого не требуется что-то особенное — годится стандартный синтаксис объявления обычных переменных. Точно так же в объявлениях переменных пользовательского типа можно использовать ключевые слова Public, Private и Static. Вот пример объявления такой переменной с помощью стандартного оператора Dim:

```
Dim usrРаботник As Персона
```

Как видите, при этом не требуется объявлять элементы внутри пользовательского типа. И хотя оператор Dim задает имя новой переменной, имена ее элементов фиксируются в объявлении соответствующего пользовательского типа.

Чтобы получить целую базу данных вашего пользовательского типа, объявите массив с помощью оператора следующего вида:

```
Private Персонал(1 To 25) As Персона
```

Обработка информации, представленной пользовательским типом данных

После объявления переменной пользовательского типа ее можно "начинять" информацией. При этом необходимо присвоить значение каждому из элементов, составляющих тип. А для этого нужно идентифицировать элемент с помощью имени переменной, за которым следуют точка и имя элемента. Например:

```
usrРаботник.strФамилия = "Елкин"
```

При заполнении нескольких элементов сразу оператор With поможет сократить объемы необходимого печатания. Вот пример использования этого оператора:

```
With usrРаботник
    .strФамилия = "Иванов"
    .strИмя = "Василий"
    .datДатаНайма = #12/9/48#
End With
```

Для присвоения данных элементам в массиве пользовательского типа используйте стандартные индексы массива. В следующем примере такой индекс задается переменной:

```
Персона(intНомер).strИмя = "Марфа"
```

Работа с переменными пользовательского типа данных

Если переменная пользовательского типа хранит полный набор данных, можно присвоить ее содержимое другой переменной того же типа— все содержимое сразу, без копания в элементах, из которых состоят эти переменные. Например, чтобы присвоить содержимое переменной usrРаботник второй ячейке в массиве Персонал, используйте оператор

```
Персонал(2) = usrРаботник
```

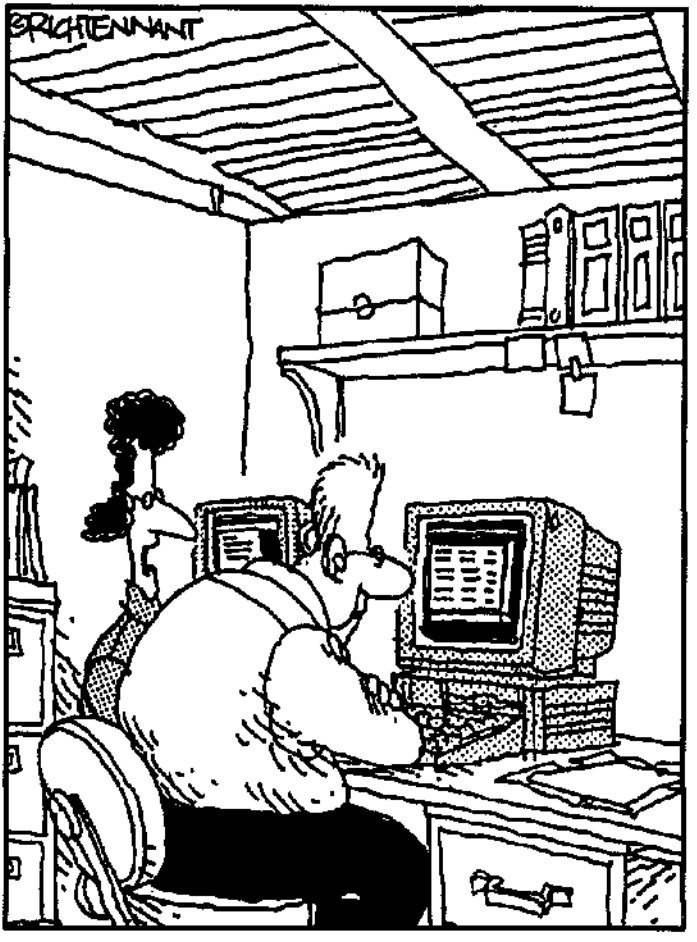
Самое интересное, что этот трюк срабатывает также в случае чтения информации из файла и записи информации в файл на диске. Например, можно сохранить полную записей базу данных на диске с помощью программного кода следующего вида:

```
Sub WriteData()
Open "C:\Database\Сотрудники.dat" For Binary As #1
For i = 1 To intРазмерБазыДанных ' цикл по всем записям
     Put #1, , Персонал(i)
Next i
Close #1 ' закрытие файла - обязательный шаг
End Sub
```

Чтение данных из файлов и запись данных в файлы на диске обсуждаются в главе 18.

Часть IV

Профессиональная работа с VBA



"Это не качественная или количественная
оценка твоей работы.
Это всего лишь скромный список пожеланий."

В этой части...

В главах с 14 по 16 рассматривается использование VBA в Microsoft Office. В главе 14 вы познакомитесь с инструментами программирования, которые работают во всех приложениях Office. Сюда относятся приемы для настройки пользовательского интерфейса, программирования Помощника по Office, который чаще всего появляется в виде Скрепыша, а также сохранения данных программы в должном формате. В главах 15 и 16 мы детально остановится на программировании для Word и Excel соответственно — наиболее широко используемых VBA-приложений. В каждой из глав подробно рассматриваются важные средства и приемы программирования для работы с документами и их содержимым с помощью VBA.

Материал остальных глав настоящей части представляет интерес для всех VBA-программистов, независимо от того, в каком приложении они работают.

В главе 17 мы поговорим о программировании для баз данных на VBA. В главе 18 мы рассмотрим не очень удобные, но более мощные не объектно-ориентированные способы работы с файлами, чем объектно-ориентированные способы, которые рассматриваются нами в главе 20. И наконец, в главе 19 я предлагаю вам расширенно рассмотреть VBA-формы.

Глава 14

VBA для Office

В этой главе...

- Отображение пользовательских панелей инструментов и кнопок с помощью VBA
- Программирование Помощника по Office
- Сохранение значений переменных на диске

VBA полезен для создания полноценных приложений не меньше, чем небольших подручных средств. В настоящей главе я познакомлю вас с некоторыми приемами, которые позволят вам профессионально работать с приложениями Office.

Контроль над панелями инструментов и меню

Приложения Office позволяют пользователям настраивать панели инструментов, строки меню и контекстные меню, обходясь безо всякого программирования, поэтому для решения подобных задач в VBA необходимости нет. Однако, если вы хотите, чтобы окно VBA-программы содержало определенный набор панелей инструментов или меню либо вам необходимо внести изменения в панели инструментов или меню при, например, запущенной программе, вам нужно написать программный код, который позволял бы в определенных условиях включать или отключать команды меню.

В последних версиях Office граница между панелями инструментов и меню стала исчезать: вы можете добавлять кнопки в меню или раскрывающиеся списка на любую панель инструментов. В VBA главная строка меню и остальные панели инструментов относятся к коллекции объектов CommandBars. Конечно же, каждая панель инструментов является отдельным объектом CommandBar.

Для работы с определенной панелью инструментов вам следует указать ее имя в коллекции CommandBars. Например, приведенный ниже фрагмент кода обращается к панели инструментов под названием VBA4Dummies:

```
Dim tbar As Toolbar
Set tbar = CommandBars("VBA4Dummies")
```

Отображение и размещение панелей инструментов

Для отображения или сокрытия панели инструментов предназначено свойство Visible. Приведенный ниже фрагмент кода отображает панель инструментов под названием VBA4 Dummies.

```
CommandBars("VBA4Dummies").Visible = True
```

Для сокрытия панели инструментов свойству Visible следует присвоить значение False.

Для переключения состояния отображения панели инструментов — отображения скрытой панели инструментов, и наоборот, следует использовать оператор Not (см. главу 11).

```
CommandBars("ReBar").Visible = _
Not (CommandBars("ReBar").Visible)
```

Если вы хотите изменить место расположения отображенной панели инструментов на экране, используйте такие ее свойства, как Position, RowIndex и Left. При необходимости используйте свойство Protection, чтобы исключить возможность перемещения панели инструментов пользователями. Подробные сведения о каждом из этих свойств приведены в следующей таблице.

Свойство	Назначение	Доступные настройки
Position	Определение того, прикреплена ли панель задач, и если так, определяется сторона, к которой она прикреплена	msoBarFloating (для неприкрепленной панели инструментов), msoBarTop, msoBarLeft, msoBarRight
RowIndex	Определение порядка закрепления панели инструментов по отношению к другим панелям инструментов, прикрепленным к той же стороне	Целое число больше 0; msoBarRowFist или msoBarRowLast (для закрепления панели инструментов первой или последней по порядку соответственно)
Left	Определяет расстояние от левого края панели инструментов до левого края экрана	
Protection	Определяет способ защиты панели инструментов от действий пользователей	msoBarNoProtection, msoBarNoCustomize, msoBarNoResize, msoBarNoMove, rasoBarNoChangeVisible, msoBarNoChangeDock, msoBarNoVerticalDock, msoBarNoHorizonalDock

Настройка кнопок панелей инструментов

Вам не потребуется VBA для создания панели инструментов, содержащей необходимый для вашей работы набор кнопок, — мощные средства настройки методом drag-and-drop, представленные в любом приложении Office, с честью справляются с подобными задачами. Но если вам необходимо изменить внешний вид панели инструментов или ее реакцию на действия пользователя буквально на лету вследствие возникновения определенных условий, VBA — именно то, что вам нужно.

Например, предположим, что вам нужно создать кнопку, которая позволит немедленно отправлять копию открытого документа Word определенному человеку или только при работе с документом определенного типа. Кнопка должна срабатывать только в определенных условиях.

Если вы знаете, как написать VBA-код, который сможет определять тип открытого документа, все, что вам необходимо, — это присвоить свойству Enabled значение True или Falseв той или иной ситуации.

Работа с определенными кнопками

Если вам необходимо настроить параметры определенной кнопки, вам необходимо идентифицировать эту кнопку в VBA-коде.

Вы можете указать кнопку одним из следующих способов.

- ✓ **Используя индексный номер кнопке в коллекции Controls объектов панели инструментов, к которой она относится.** Индексный номер кнопки соответствует ее расположению на панели инструментов; при этом кнопке, расположенной в крайнем левом (или в крайнем верхнем) углу, соответствует индексный номер 1. Ниже приведен пример подобного обращения к объекту:

```
CommandBars("BarNone").Controls(3)
```

  Проблема состоит в том, что Office позволяет пользователям перетаскивать кнопки по панели инструментов, поэтому ничто не гарантирует, что вы обращаетесь именно к той кнопке, которая вам необходима.

- ✓ **Использование свойства Caption кнопки. Вы** можете ввести любой заголовок, используя пользовательский интерфейс Office. Выберите команду Сервис⇨Настройка, перейдите на вкладку **Команды** появившегося диалогового окна **Настройка.** Щелкните на интересующей вас кнопке правой кнопкой мыши, чтобы увидеть имя кнопки. Следующая команда ссылается на кнопку, свойство Caption которой равно "Угадай мой вес".

```
CommandBars("BarNone").Controls("Угадай мой вес")
```

  Вы можете определить заголовок прямо в программном коде, но вам все равно придется указать кнопку одним из других способов.

- ✓ **Используя метод FindControl коллекции CommandBars для обнаружения кнопки.** Этот метод обнаруживает кнопки по нескольким критериям. Наиболее полезным оказывается использование свойства Tag. Однако, чтобы оно сработало, вам сначала необходимо присвоить этому свойству уникальное значение, используя инструкцию, подобную приведенной ниже.

```
CommandBars("BarNone").Controls(,,"Tag01")
```

  После этого, когда вам необходимо обратить к кнопке из программного кода, вы можете использовать метод FindControl для получения ссылки на кнопку, например, с помощью следующей инструкции:

```
CommandBars.FindControl(,,"Tag01")
```

В этом примере запятые относятся к необязательным параметрам метода FindControl, которые не потребуются вам при нахождении элементов управления (в нашем случае, кнопки) по свойству Tag.

Настройка кнопок на лету

В приведенной ниже таблице перечислены наиболее полезные свойства и методы, позволяющие настраивать кнопки панелей инструментов с помощью VBA,

Свойство или метод	Эффект	Доступные параметры (только для свойств)
Свойство Enabled	Определяет, включена ли кнопка и доступна ли она щелчком мышью либо она отключена, а значит, затенена	True или False

Свойство или метод	Эффект	Доступные параметры (только для свойств)
Свойство Visible	Определяет, видна ли кнопка	True или False
Свойство State	Определяет, кажется ли кнопка "нажатой"	msoButtonUp (обычный вид кнопки); msoButtonDown (кнопка заблокирована в нажатом состоянии); msoButtonMixed (кнопка имеет вид "нажимаемой")
Метод CopyFace	Копирует изображение с указанной кнопки в буфер обмена. Для кнопки, изображение которой изменяется должным образом, создайте панель инструментов, содержащую кнопки с нужным изображением, но не выполняющие никаких реальных действий. Затем скройте эту панель инструментов. Когда придет время изменить изображение на "настоящей" кнопке, используйте метод CopyFace для копирования необходимого изображения с соответствующей кнопки на скрытой панели инструментов. Затем используйте метод PasteFace для вставки изображения на кнопку	
Метод PasteFace	Размещение на кнопке изображения, уже помещенного в буфер обмена	

Отображение пользовательских экранных подсказок для кнопок на панели инструментов

Экранные подсказки — это те небольшие сообщения желтого цвета, которые появляются на экране после подведения указателя мыши к определенному элементу на одну-две секунды. Экранные подсказки для панелей инструментов должны быть включены, и по умолчанию это именно так. Если же это не так, выберите команду **Сервис⇨Настройка**, перейдите на вкладку Параметры, после чего установите флажок Отображать подсказки для кнопок.

По умолчанию экранная подсказка для кнопки содержит только ее имя. Однако, если вы не боитесь немного попрограммировать на VBA, вы сможете добавить текст к экранной подсказке, изменив ее свойство TooltipText. Приведенный ниже программный код добавляет экранную подсказку для воображаемой кнопки в Excel:

```
Sub Этой_Кнопке_Нужна_Экранная подсказка!)
CommandBars("Mr. GoodBar").Controls(2).ToolTipText = _
    "Форматирование ячейки: по центру, полужирный, денежный"
End Sub
```

Вам необходимо выполнить эту процедуру всего один раз. Поскольку Office сохраняет новые экранные подсказки в шаблоне документа, с которым вы работаете, вы можете удалить процедуру в любое время.

Программирование Помощника по Office

В любом приложении из состава Microsoft Office Помощник по Office позволяет быстро находить необходимую справочную информацию или предлагает пошаговые инструкции для выполнения повседневных задач. Как вы и могли ожидать, для манипулирования Помощником с помощью VBA достаточно использовать объект `Assistant`.

Контроль над Помощником

Используйте свойства объекта `Assistant` (табл. 14.1) для контроля за тем, как и когда Помощник по Office будет появляться на экране. Пока вы не уверены в том, что Помощник уже включен, убедитесь в том, что его свойство On имеет значение `True`, прежде чем пытаться включить его снова.

Таблица 14.1. Свойства объекта Assistant

Свойство	Функция	Допустимые значения
On	Определяет доступность Помощника по Office. Соответствует флажку Использовать помощника на вкладке Параметры диалогового окна Помощник	`True` или `False`
`Visible`	Определяет наличие Помощника на экране	`True` или `False`
`FileName`	Выбирает персонаж помощника, который будет представлен на экране: Скрепыш, Мурка и т.д.	Допустимое имя файла с расширением `.acs`, заключенное в кавычки
`Animation`	Заставляет Помощника выполнять различные трюки. Вы можете указать, какие именно действия будет выполнять Помощник	Любая из доступных констант, определяющих конкретные движения, такие как `msoAnimationGetArsty`. Для просмотра списка всех доступных констант выберите тип `MsoAnimationType` в библиотеке Office в окне обозревателя объектов

Программирование окна Помощника

Окно Помощника — это тот желтый прямоугольник, в котором Помошник предлагает вам возможные варианты действий. Хотя Office использует это окно практически исключительно для отображения справочной информации, вы сможете использовать VBA для создания собственных вариантов окна и заполнения его всевозможными сведениями, такими как котировки акций, процентные ставки банков и т.д.

Создание окна

Первый шаг к отображению пользовательского варианта окна Помощника состоит в создании нового объекта. Как это ни удивительно, но `NewBaloon` — это *свойство*, а не метод объекта `Assistant`. В любом случае, используйте инструкцию `Set` для создания объекта окна с помощью такого кода, как, например, приведен ниже.

```
Dim blnUpUpAndAway As Baloon
Set blnUpUpAndAway = Assistant.NewBaloon
```

После создания объекта balloon используйте перечисленные в табл. 14.2 свойства для определения его содержимого. В нашем случае используйте метод Show для отображения пользователю окна Помощника.

Таблица 14.2. Свойства объекта Balloon

Свойство	Для чего предназначено	Доступные параметры; заметки
Mode	Определяет, должен ли пользователь закрыть окно Помощника, чтобы продолжить работу с приложением	msoModeModeless (окно остается на экране до тех пор, пока не будет закрыто пользователем); msoModeModal (окно должно быть закрыто пользователем); rasoModeAutoDown (окно закрывается после щелчка за его пределами)
BalloonType	Определяет, должны ли заголовки в окне быть в виде кнопок или же в виде нумерованного или маркированного списка	msoBalloonTypeButtons; msoBalloonTypeNumbers; msoBalloonTypeBullets
Icon	Указывает на один из доступных значков для окна Помощника	msoIconNone;msoIconTip; msoIconAlert
Heading	Определяет заголовок, отображаемый полужирным шрифтом в верхней части окна Помощника	Любое строковое значение
Text	Определяет "обычный" текст, который отображается после заголовка, но над любыми подписями, флажками или кнопками	Любое строковое значение
Labels	Указывает текст подписей в окне Помощника	Любое строковое значение
CheckBoxes	Указывает текст подписей к флажкам в окне Помощника	Указывает текст подписей в окне Помощника
Button	Определяет количество и тип кнопок, отображаемых в нижней части окна Помощника	Любая из констант MsoButtonSetType, перечисленных в окне обозревателя объектов

Добавление графических изображений

Вы можете придать окну Помощника гораздо более привлекательный внешний вид, если добавите в него хотя бы одну картинку. Для отображения картинки вам следует присвоить свойству Icon значение, равное одной из констант, перечисленных в табл. 14.2. Для отображения картинки определите ее как часть текстовой строки, используемой для определения свойств Heading или Text объекта окна Помощника, либо одной из подписей, как показано в следующем примере:

```
blnUpUpAndAway.Text = "VBA! {bmp c:\dummies.bmp}"
Как видно из этого примера, вы должны заключить имя файла
```

с изображением и путь к нему в фигурные скобки. Сначала вам необходимо указать формат файла — bmp для формата .bmp или wmf для формата .wmf. и только затем указывают путь к файлу и имя файла.

Закрытие окна

После того как созданное вами окно Помощника отображено на экране, вам обязательно потребуется убрать его с экрана. Если объект окна Помощника является модельным (его свойству Mode присвоено значение msoModeModal, что имеет место по умолчанию), окно будет закрыто после щелчка на любой из его кнопок. Если же это не так, вам необходимо использовать в своей процедуре метод Close для закрытия окна в результате наиболее вероятных действий со стороны пользователя — щелчка на кнопке OK или на кнопке Отмена.

Работа с элементами управления окна

Точно так же как и в обычных формах VBA, элементы управления в окне Помощника предоставляют пользователю возможность взаимодействия с программой. Окно Помощника может содержать элементы управления трех типов: флажки, надписи (могут выступать в качестве кнопок) и кнопки (это кнопки, которые располагаются в нижней части окна Помощника).

Добавление надписей и флажков

Каждое окно Помощника содержит пять надписей и пять флажков. Однако, до тех пор пока вы не укажете текст для определенной надписи или флажка, соответствующий элемент управления просто не отображается в окне. Для изменения значения свойства Text надписи или флажка вам следует обратиться к нему по его номеру в соответствующей коллекции объектов, как показано ниже:

```
With blnUpUpAndAway
     .Labels(2) = _
          "Не забудьте приобрести молоко по пути домой."
     .Checkboxes(3).Text = "Я уже приобрел молоко."
End With
```

Надписи могут действовать в роли кнопок, реагируя на щелчки кнопкой мыши, но только в том случае, если вы присвоите свойству BalloonType объекта окна Помощника значение msoBalloonTypeButtons. Другие значения этого свойства позволяют представлять надписи в виде нумерованных или маркированных списков (см. табл. 14.2).

Добавление кнопок

Вы указываете количество кнопок и их тип, присваивая свойству Button объекта окна Помощника значение, равное одной из констант типа msoButtonSetType. Например, если вы хотите, чтобы окно Помощника содержало три кнопки: Abort (Прервать), Retry (Повтор) или Ignore (Игнорировать), ваш код должен иметь следующий вид:

```
blnUpUpAndAway.Button = msoButtonSetAbortRetryIgnore
```

Считывание значений кнопок и флажков

Программисту просто необходим способ определения того, на какой кнопке щелкнул пользователь или какой флажок он установил. Вот как это делается: для определения того, на какой именно кнопке щелкнул пользователь, используется значение, возвращаемое методом Show: оно предоставляет вам число, представляющее кнопку. Вы можете присвоить это значение переменной или использовать его напрямую, как показано в следующих примерах:

```
intButton = blnUpUpAndAway.Show

Select Case blnUpUpAndAway.Show
```

```
      Case msoBalloonButtonOK
            (код, выполняемый после щелчка на кнопке OK)
      Case msoBalloonButtonCancel
            (код, выполняемый после щелчка на кнопке Cancel)
      Case 2
      (код, выполняемый после щелчка на кнопке со другой надписью]
End Select
```

Как видно из двух первых инструкций `Case`, библиотека объектов `Office` содержит именованные константы, которые соответствуют значениям каждого типа "обычных" кнопок (т. е. кнопок, которые отображаются в нижней части окна Помощника). Вы можете использовать эти константы для определения того, на какой именно из кнопок щелкнул пользователь. Для определения щелчка на кнопке просто используйте номер подписи к кнопке, как показано на примере третьей инструкции `Case`.

Для считывания сведений о текущем состоянии флажков в окне Помощника используйте свойство `Checked` каждого из них после закрытия окна пользователем. В самом простейшем случае вы можете использовать последовательность операторов `If ... Then`, по одному оператору для каждого флажка, как показано ниже:

```
With blnUpUpAway
      If .Checkboxes(1).Checked Then
            (код, выполняемый в том случае, если флажок 1 установлен)
      Else
            (код, выполняемый в том случае, если флажок 1 не установлен)
      End If
If .Checkboxes(2).Checked Then
            (код, выполняемый в том случае, если флажок 2 установлен)
      Else
            (код, выполняемый в том случае, если флажок 2 не установлен)
      End If
End With
```

Вы можете использовать свойство `Callback` объекта `Balloon` для определения процедуры, которая будет обрабатывать щелчки на кнопках. Этот прием оказывается особенно полезным при использовании немодальных объектов `balloon`, которые остаются на экране до тех пор, пока пользователь не щелкнет на кнопке.

Сохранение значений переменных на диске

В программировании очень часто возникают ситуации, когда необходимо сохранить значения переменных во время выполнения программы. Например, давайте рассмотрим простейший случай: использование переменной-счетчика. Предположим, вам необходимо знать, сколько раз пользователь щелкнул на определенной кнопке одной из панелей инструментов Excel на протяжении месяца. Если вы только не хотите держать программу Excel постоянно запущенной на протяжении целого месяца, вам необходимо сохранять текущее значение переменной-счетчика при закрытии программы и считывать его при последующем запуске Excel.

Среди всех приложений Office только Word предоставляет вам инструменты для хранения подобных переменных (с помощью объекта `Variable`). Однако с подобной задачей по силам справиться и многим другим приемам.

Сохранение и получение значений переменных в Excel, PowerPoint и Project

Пользовательские свойства документов предоставляют широкое поле для деятельности при написании программ для большинства приложений Office. Вы можете использовать VBA для создания пользовательских свойств документов, заполнения их данными, а также получения этих данных при первой необходимости. Раз подобные свойства стали неотъемлемой частью документа, с которым они связаны, вам не придется беспокоиться о сохранении свойств в отдельном файле — они автоматически сохраняются вместе с документом.

Для создания нового свойства документа в Excel, PowerPoint или Project (в Word тоже при необходимости можно воспользоваться подобным методом) выберите метод Add из коллекции свойств CustomDocuraentProperties этого документа. Приведенный ниже пример срабатывает в Excel, где объект ActiveWorkbook означает активную рабочую книгу во время выполнения процедуры (в Word и PowerPoint это объекты ActiveDocument и Active Presentation соответственно):

```
ActiveWorkbook.CustomDocumentProperties.Add_
     Name:= "Button Count", LinkToContent:=False,_
     Type:= msoPropertyTypeNumber, Value:=0
```

Параметр LinkToContent должен иметь значение False, если вы сохраняете значение переменной в виде пользовательского свойства документа (вам следует присвоить этому параметру значение True, если вы хотите, чтобы значение свойства было связано с содержимым ячейки Excel или закладкой Word). Параметр Type определяет тип данных для свойства; в качестве значения допустимо использование таких констант, как msoPropertyTypeNumber, msoPropertyTypeBoolean, msoPropertyTypeDate, msoPropertyTypeFloat или msoPropertyTypeString.

Для сохранения или получения информации, хранящейся в виде свойства документа, используется свойство Value объекта Property. Предположим, что вы используете такое свойство документа, как ButtonCount, для сохранения количества щелчков мышью на кнопке одной из форм программы. Приведенный ниже фрагмент кода считывает и обновляет значение свойства после каждого щелчка на кнопке:

```
With ActiveWorkbook._
     CustomDocumentProperties("ButtonCount")
          .Value = .Value + 1
End With
```

Приведенный ниже пример иллюстрирует выполнение небольшой практической задачи. В этом случае пользователь видит сообщение о том, что он щелкнул на кнопке больше определенного количества раз, даже в том случае, если между щелчками документ неоднократно закрывался и открывался заново. Предположим, что значение константы cintCutOffValue задано раньше равным минимальному количество щелчков, после достижения которого сообщение отображается на экране в первый раз.

```
With _
ActiveWorkbook.CustomDocumentProperties("ButtonCount")
     'получение сохраненного значения свойства
     intCurrentCount = .Value
     'отображение сообщения при выполнении условия
     If intCurrentCount > cintCutOffValue Then
          MsgBox "Вы щелкнули на кнопке" _
               & " больше " & cintCutOffValue _
               & " раз."
```

```
    End If
    'Увеличение значения свойства
    .Value == intCurrentCount + 1
End With

intCurrentCount = ActiveWorkbook
_.CustomDocumentProperties("ButtonCount").Value
```

Другие способы сохранения значений переменных

В Access и Excel часто удобно сохранять данные, необходимые для работы VBA-программы, прямо в документе — в таблицах базы данных (Access) или ячейках листа рабочей книги (Excel). Единственная проблема, связанная с подобным подходом, состоит в том, что данные видны пользователям и их легко изменить, если только вы не предпримите предупреждающих действий.

Два остальных возможных решения для сохранения данных на диске сработают в любом приложении, поддерживающем VBA. Вы можете сохранять значения в реестре Window (и даже считывать их оттуда), используя инструкции SaveSetting и GetSetting, а также сохранять данные в отдельном файле. Оба эти способа рассмотрены в главе 20.

Глава 15

Программирование на VBA в Word

В этой главе...

- Объектная модель Word
- Ключевые объекты Word: окна, выделения, диапазоны и объект Find
- Диапазоны или выделения: что использовать при работе с текстом
- Методы и свойства для манипуляции с текстом
- Поиск и замена текста с помощью VBA в Word

Из всего набора приложений Office Word предлагает наиболее богатый набор средств программирования. Знакомство с объектами Word, такими как Range и Find, иногда оказывается рискованным, но это очень важно, если вы действительно хотите создавать VBA-процедуры, раскрывающие всю мощь Word. Именно об этом мы и поговорим в настоящей главе.

Объектная модель Word содержит так много объектов и их коллекций, что вам потребуется огромный лист бумаги, если вы захотите построить на нем иерархию всех этих объектов в графическом виде. Очевидно, что я смогу рассмотреть лишь малую толику от того колоссального количества объектов, свойств и методов, которые вы сможете использовать в своих программах. В настоящей главе освещены только наиболее важные приемы работы с Word VBA. Если вы хотите получить более подробные сведения, вам придется неоднократно обращаться к справочной системе.

Знакомство с объектом Application

Как и во всех других VBA-приложениях, ключевым в объектной модели Word является объект Application. Другими словами, этот объект содержит все остальные объекты Word. Поскольку объект Application занимает центральное место в программировании на VBA в Word, вам даже не придется явно указывать его имя при работе со многими важными объектами. Однако вам не следует забывать о роли, которую играет этот объект, поскольку он вам потребуется при работе со свойствами и методами самого приложения, а также при обращении к некоторым другим объектам. Например, приведенная ниже инструкция использует метод ListCommands объекта Application:

```
Application.ListCommands (True)
```

Между прочим, метод ListCommands создает новый документ и помещает в него таблицу, содержащую комбинации клавиш и команды меню Word. Если вы передаете методу ListCommands значение True, новый документ содержит комбинации клавиш и команды меню Word. Передайте ему значение False, чтобы в нем перечислялись только команды.

Доступ к документам Word с помощью VBA

Если написанная вами VBA-процедура функционирует прямо в документе, вы должны указать объект этого документа непосредственно в коде, Часто вы сможете сделать это неявным образом, используя объект Selection, о котором я расскажу дальше в настоящей главе. Однако в остальных ситуациях вам придется явно идентифицировать целевой объект.

Работа с активным документом

Типичная VBA-процедура в Word выполняет все свои магические действия в том документе, который в данный момент открыт для редактирования. Для указания *активного документа* используется объект ActiveDocument. Например, приведенная выше инструкция просто закрывает активный документ:

```
ActiveDocument.Close
```

Как видите, вам не нужно писать код для определения того, какой же документ редактируется в данный момент: достаточно просто использовать объект ActiveDocument.

Указание конкретного документа

Если вам необходимо работать с определенным документом, который сейчас не активен, вы должны указать его как члена коллекции Documents, состоящей изо всех документов, открытых в настоящее время в Word. Как и в случае с любой другой коллекцией объектов в VBA, вы можете обратиться к отдельному документу в коллекции, используя его заголовок, который в данном случае совпадает с именем файла (только именем файла, а не с полным путем к нему). Вот соответствующий пример;

```
Documents("Toy Store Newsletter.doc")
```

Поскольку вы не знаете точно имя файла целевого документа заранее, а пользователь может в любой момент его изменить, вам следует создать переменную, которая будет содержать имя файла. После этого вы можете использовать эту переменную для указания объекта документа, например, так: Documents(strDocName).

Вы также можете обратиться к документу по его индексному номеру. Следующая инструкция, например, обращается к третьему документу в коллекции Documents:

```
Documents(3)
```

Несмотря на всю простоту, этот метод достаточно ограничен, поскольку вы редко когда знаете индексный номер документа, с которым хотите работать. При этом вам стоит узнать имя открытого документа. Например, следующая инструкция присваивает переменной имя файла второго открытого документа:

```
strDocName = Documents(2).Name
```

Создание, открытие, активизация и закрытие документов

Для создания нового документа используйте метод Add коллекции объектов Documents. Используемый без каких-либо документов, метод Add создает новый документ, базирующийся на шаблоне Normal (Обычный). Для указания другого шаблона укажите путь к нему в качестве аргумента, как показано ниже:

```
Documents.Add template:= _
"C:\Windows\Application Data\Microsoft\Hidden templates"
```

Для открытия существующего документа используйте метод Open коллекции объектов Documents. Конечно же, вам необходимо указать полный путь к документу, как показано ниже:

```
Documents.Open FileName:= _
     "C:\Toys\Toys for infants.doc"
```

Для активизации уже открытого документа используйте метод Activate коллекции объектов Documents. Предположим, что вам необходимо, чтобы ваша VBA-программа активизировала определенный документ, который в момент запуска программы может быть и открыт, и закрыт. Используйте код, подобный показанному ниже, для активизации открытого документа или открытия документа, если он еще не открыт:

```
Sub DocActivateOrOpen()
Dim docFileName As String, docPath as String
docFileName = "Старые игрушки.doc"
docPath = "C:\Toys\"
For Each targetDoc In Documents
     If targetDoc.Name = docFileName Then
          targetDocIsOpen = True
     End If
Next targetDoc
If targetDocIsOpen = True Then
     Documents(docFileName).Activate
Else
     Documents.Open FileName := docPath & docFileName
End If
End Sub
```

Работа с разделами документов

Поскольку каждый документ Word состоит из одного или нескольких разделов, вы можете ожидать, что Word VBA содержит коллекцию Sections и отдельные объекты Section для работы с этими элементами. Так оно и есть. Наиболее важное применение объектов Section — организация доступа к верхним и нижним колонтитулам (через объект HeaderFooter). Вы можете добавлять новые разделы в документ, используя метод Add коллекции Sections или метод InsertBreakобъекта Range или Selection.

Открытиокон с помощью VBA

Каждый открытый документ содержит как минимум одно окно, а пользователь Word может открыть столько окон, сколько нужно для работы с любым документом. Каждое из подобных окон — объект с его собственными правами. В объектной модели Word объект Application содержит коллекцию Windows, содержащее все окна для всех открытых документов. Кроме того, каждый объект Document содержит свою собственную коллекцию объектов Windows.

Две основные причины работы с объектами Window в Word — контроль над внешним видом окна, а также манипулирование содержимым документа с помощью объекта Selection. Объект Selection я подробно рассмотрю в следующем разделе настоящей главы; здесь же я остановлюсь на приеме, позволяющем определить конкретное окно, а также познакомлю со свойствами, которые вы сможете использовать для определения внешнего вида окна.

Обращение к окнам из программного кода

Прямо из программного кода проще всего обращаться к тому окну, в котором открыт редактируемый документ во время запуска процедуры. Для указания окна используется объект ActiveWindow.

Для определения окна прямо в программном коде вам следует определить его как члена одной из коллекций Windows. При работе с глобальной коллекцией объектов Windows вам не нужно указывать собственно объект Application. При обращении к коллекции объектов Windows определенного документа, указывать имя объекта этого документа нужно обязательно. Вы можете идентифицировать окно по его имени или индексному номеру в коллекции. Имя окна совпадает с именем документа, который в нем отображается, за исключением того, что если для одного документа открыто несколько окон, после указания имени документа следует ставить точку с запятой, после которой указать номер окна.

Ниже приведены типичные ссылки для объектов Window.

Ссылка	Комментарии
`Windows("Document4")`	Действительна, если для документа Document4 открыто только одно окно
`Windows("Kites and skip ropes.doc:3")`	Указывает на третье окно документа
`Documents("Window display.doc").Windows(2)`	Указывает на второе окно в коллекции Windows данного документа

Работа с областями окон

Когда вы разделяете окно с помощью команды Окно⇨Разделить, верхняя и нижняя часть окна являются отдельными *областями*. Окно Word содержит как минимум одну область, но может содержать и больше областей. В областях Word также отображает и такие сведения, как верхние и нижние колонтитулы, сноски и комментарии.

Если вам необходимо получить доступ к настройкам внешнего вида выделения в отдельной области, вы должны сначала идентифицировать в программном коде целевой объект Pane. Обращение к определенному объекту осуществляется по его номеру в коллекции объектов Panes для данного окна. Однако вы можете опустить ссылки на области в том случае, когда вам необходимо работать со стандартной основной частью окна (или верхней частью, если окно разделено).

Изменение внешнего вида окна

Объекты Window предлагают большое количество свойств, определяющих состояние всех элементов, которые вы видите на экране. Основная часть этих свойств может принимать только одно из двух возможных значений — True или False. Например, чтобы включить схему документа для активного документа, используйте следующую инструкцию:

```
ActiveWindow.DocumentMap = True
```

Используйте подобные инструкция для включения или выключения различных свойств, таких как DisplayScreenTips или DisplayVerticalScrollBar. Не забывайте о том, что ключевое слово Not обращает текущее значение логической переменной или свойства. Вот пример изменения значения свойства:

```
ActiveWindow.DisplayRules = _
    Not ActiveWindow.DisplayRules
```

Свойства Left,Top, Height и Width позволяют вам определять размер и расположение окна, которое не развернуто на весь экран.

Использование объекта View

Вспомогательный объект View определяет многие аспекты отображения окна или области. Объект View обладает следующими свойствами.

Свойство объекта View	Для чего предназначено
Type	Соответствует выбору из меню Вид таких команд, как Обычный, Разметка страницы, Структура и т. д. Для изменения представления используйте в качестве значения свойства одну из следующих констант: wdMasterView, wdNormalView, wdOutlineView, wdPrintView, wdWebView или wdPrintPreview. Например, инструкция ActiveWindow.View.Type = wdPrintView переключает активное окно в режим предварительного просмотра
FullScreen	Контролирует отображения окна в стандартном или полноэкранном варианте
TableGridlines	Определяет отображение сетки таблицы
ShowAll, Show	Свойство ShowAll определяет отображение всех непечатаемых знаков, что соответствует флажку Все в группе Знаки форматирования на вкладке Вид диалогового окна Параметры. Вы можете включить или отключить отображение отдельных непечатаемых символов, а также других элементов, таких как выделение текста или границы, используя различные свойства, названия которых начинаются с Show, в частности ShowBookmarks или ShowHighlight

Масштабирование документа с помощью программного кода

Для управления параметрами увеличения документа вам необходимо немного углубиться в иерархию объектов, чтобы добраться до объекта Zoom, а затем изменить значение его свойства Percentage. Соответствующий пример приведен ниже:

```
ActiveWindow.View.Zoom.Percentage = 135
```

Если вы хотите задать новый масштаб документа для представления, которое в данный момент не используется, включите константу в качестве аргумента свойства View для этого представления, как показано ниже:

```
ActiveWindow.View.Zoom.Percentage = 75
```

Когда пользователь в следующий раз выберет соответствующее представление документа, последний сразу предстанет в указанном вами масштабе.

Кроме того, вы можете использовать свойство PageFit объекта Zoom для дублирования команд По ширине страницы или Страница целиком из раскрывающегося меню Масштаб на панели инструментов. При активной любой из этих настроек Word автоматически изменяет масштаб документа при изменении размеров окна, всегда сохраняя указанную вами пропорцию. Например, следующая инструкция эквивалентна команде По ширине страницы из раскрывающегося меню Масштаб на панели инструментов:

```
ActiveWindow.View.Zoom.PageFit = wdPageFitBestFit
```

Команда Страница целиком доступна только в режиме разметки страницы, но вы можете продублировать ее с помощью приведенного ниже кода:

```
Windows("Document1").View.Zoom.PageFit = _
    wdPageFitFullPage
```

Для отключения функции автоматического масштабирования присвойте свойству PageFit значение wdPageFitNone.

Использование объекта Selection

В Word VBA объект Selection означает любой выделенный элемент в области окна. Да-да, объект Selection относится к области окна, а не документа. Документ может содержать больше одного открытого окна, а каждое окно — несколько областей, поэтому каждая из этих областей может содержать выделения. (Хотя чисто технически объект Selection принадлежит области окна, ничто не мешает вам считать его принадлежащим окну, если только вам не нужно работать с выделением в определенной области окна, например в нижнем или верхнем колонтитуле).

Хотя вы можете манипулировать выделенными областями с помощью VBA-кода, используя объект Selection, часто гораздо удобнее оказывается использовать объект Range. Подробно об использовании этих объектов мы поговорим в разделе "Работа с текстом в Word VBA" дальше в настоящей главе.

Содержимое выделенной области может быть фрагментом текста, таблицей, текстовым окном, изображением или чем-нибудь другим, что можно выделить с помощью мыши и клавиатуры. Запомните: если ничего не выделено, объект Selection представляет текущее расположение точки вставки.

Хотя каждая область окна содержит выделение, вам необходимо только явно обратиться к целевому окну, если интересующее вас выделение *не находится* в главной области активного окна. Для работы с выделенной частью в главной области активного окна вам следует использовать объект Selection. Например, вы можете использовать приведенную ниже инструкцию для замены выделения в области активного окна текстом, заключенным в кавычки:

```
Selection.Text = "У моей собаки есть блохи"
```

Если вам необходимо работать с выделенной областью в одном из неактивных окон, вы должны полностью указать имя окна:

```
Documents("Songs.doc").Windows(2).Selection.Text = _
    "Моя возлюбленная загорает на берегу океана"
```

Поскольку объект Selection может представлять содержимое различных типов, всегда лучше всего сначала проверить, данные какого вида были выделены, прежде чем с ними что-то делать. В противном случае вы рискуете получить неожиданные результаты или даже сообщения об ошибках. Для поиска подобных сведений используйте свойство Type объекта Selection. Например, приведенный ниже фрагмент кода проверяет, является ли выделенная область обычным текстом, прежде чем вырезать ее в буфер обмена:

```
With Selection
If .Type = wdSelectionNormal Then
    .Cut
End If
```

Вы можете использовать при подобных проверках константы, перечисленные ниже.

Константа	Что выделено
wdNoSelection	Ничего не выбрано
wdSelection3lock	Вертикальный блок текста
wdSelectionColumn	Столбец в таблице
wdSelectionFrame	Рамка
wdSelectionInlineShape	Графическое изображение в тексте
wdSelectionIP	Только точка вставки
wdSelectionNormal	Обычное выделение фрагмента текста
wdSelectionRow	Строка таблицы
wdSelectionShape	Изображение, не помещенное в текст

Знакомство с объектами Range

Если вы редактируете документ самостоятельно, вы должны поместить указатель мыши в нужное место либо выделить определенный фрагмент, прежде чем добавлять, удалять или форматировать текст. Однако в Word объекты Range избавляют вас от подобной необходимости. Объект Range определяет неразрывный блок текста в документе. Объекты Range полностью независимы от точки вставки или выделенного раздела, которые пользователь видит в окне документа. После создания объекта Range вы сможете манипулировать текстом, используя команды VBA, полные аналоги мощных команд редактирования Word, точно так же, как при использовании объектов Selection.

Вы можете определить объекты Range в программном коде одним из двух следующих способов:

- ✓ работая с диапазонами с помощью свойства Range;
- ✓ определив диапазоны с помощью метода Range объекта Document.

Использование свойства Range

Открытый документ Word уже содержит объекты Range, соответствующие многим его элементам. Каждый абзац, а также таблица, отдельная ячейка таблицы, комментарий или нижний колонтитул (и это далеко не полный список) определяют диапазоны. Вы можете считать, что все эти диапазоны существуют в некотором виртуальном мире, пока не вы не обратитесь к ним с помощью свойства Range соответствующего объекта. Например, для определения объекта Range, соответствующего первому абзацу активного документа, вам следует использовать объектную ссылку следующего вида.

```
ActiveDocument.Paragraphs(1).Range
```

Поскольку подобные стандартные диапазоны уже существуют в Word, вы можете использовать объектные ссылки на них напрямую, обходясь без каких-либо переменных. Это особенно хорошо подходит для тех ситуаций, когда вам необходим один конкретный диапазон для одной операции. Например, приведенная ниже инструкция копирует вторую таблицу в документе в буфер обмена, используя метод Copy объекта Range:

```
ActiveDocument.Tables(2) .Range.Copy
```

Если несколько инструкций используют один и тот же диапазон, вы можете использовать конструкцию With... для ускорения как ввода кода программы, так и ее выполнения. Ниже приведен пример кода, который проводит сортировку абзацев в третьем разделе документа, отображая первое предложение полужирным:

```
With ActiveDocument.Section(3).Range
     .Sort SortOrder := wdSortOrderAscending
     .Sentences(1).Range.Bold = True
End With
```

Приведенный выше пример иллюстрирует, как объект Range содержит другие объекты, в свою очередь содержащие диапазоны. Инструкция в третьей строке обращается к дипазону, соответствующему первому предложению в исходном диапазоне, после чего отображает его полужирным. Также обратите внимание на то, что вы не можете применить форматирование непосредственно к объектам, таким как слова, предложения или абзацы; для того чтобы это сделать, вам следует использовать их свойства Range.

Если вы планируете использовать диапазон в нескольких инструкциях, которые не следуют одна за другой, вам придется присвоить диапазон переменной. Это позволит вам быстрее составить код программы, а также в конечном итроге ускорить ее выполнение.

Объекты Selection также обладают свойством Range. Это значительно упрощает использование свойств и методов, принадлежащих объектам Range существующих выделенных областей. Приведенный ниже пример присваивает диапазон выделенной области переменной, перемещает выделенную область, после чего преобразует текст прописными буквами:

```
Set deRange = Selection.Range
Selection.Move Unit := wdParagraph, Count := 3
deRange.Case = wdLowerCase
```

Определение диапазонов с помощью метода Range

Если существующие объекты не содержат текст, с которым вы хотите работать, создайте собственный объект Range. В любом открытом документе вы сможете определить столько объектов Range, сколько вам нужно. При этом используется метод Range документа, который требует от вас указания начальной и конечной точек диапазона в терминах расположения символа в документе. Например, ознакомьтесь со следующим примером:

```
ActiveDocument.Range(Start:= 10, End:=20)
```

Приведенное выше выражение представляет собой объектную ссылку на диапазон, который начинается с 11-го символа и заканчивается 20-м символом. Значение расположения символа на самом деле означает место слева от данного символа. Например, значение, равное 0, соответствует первому символу в документе, а значение 10 указывает на точку между 10-м и 11-м символами. Word считает *все* символы в документе, включая скрытые и непечатаемые знаки.

Для создания диапазона, который станет всего лишь местом расположения и не содержит никакого текста, присвойте начальному и конечному значениям, определяющим диапазон, одно и то же число. Для включения в объект Range всего документа используйте метод Range этого документа без каких-либо аргументов или же используйте свойство Content документа.

Объект Range создать совсем несложно (если вам известны положения начального и конечного символов, которые вы решили включить в диапазон). Проблема возникает в том случае, когда вам необходимо работать с произвольным количеством символов в произвольном месте документа. Чаще всего вам необходимо работать с текстом в определенной части документа. Вы можете начать диапазон с существующей закладки, с начала выделенной области или с определенного слова или фразы, о наличии которых в документе вам точно известно.

Для определения диапазона, базирующегося на одном из подобных критериев, используйте свойства Start или End объекта Selection, Range или Bookmark, чтобы определить месторасположение в документе интересующего вас символа. Если вам необходимо создать состоящий из десяти символов диапазон, который будет начинаться с вкладки ForgetMeNot, вам пригодится следующая инструкция;

```
With ActiveDocument
Set myEkMark = .Bookmarks("ForgretMeNot")
Set homeOnTheRange =
     .Range(Start := myBkMark, End := myBkMark + 10)
End With
```

Ниже приведен еще один пример, иллюстрирующий использование свойства Range для поиска абзаца, определения в нем некоторого слова, а также создания нового диапазона, начинающегося с этого слова. В данном случае аргумент End опущен, поэтому диапазон начинается с указанного слова и продолжается до окончания документа:

```
With ActiveDocument
     Set fistWord = .Paragraphs(160).Range.Words(3)
     Set RangeTop = .Range(Start := firstWord.Start)
End With
```

В разделе "Поиск и замена текста с помощью VBA в Word", дальше в настоящей главе, я покажу, что использование ключевого слова Find совместно с диапазоном или выделенной областью определяет объект, который будет содержать только найденный текст. После того как инструкция Find найдет в диапазоне или выделенной области фразу, свойства Start и End того же диапазона или выделенной области теперь будут указывать на начало и конец найденного текста.

Работа с текстом в Word VBA

Объекты Range и Selection являются отправными точками для практически любых операций, которые вы сможете выполнять с текстом с помощью Word VBA. Некоторые из этих действий можно применять к документам в целом, но в общем случае вам необходим диапазон или выделенная область, прежде чем вносить изменения.

У объектов Range и Selection достаточно много общего, но есть и несколько ключевых отличий. Оба объекта представляют непрерывные последовательности символов, над которыми вы сможете выполнять различные операции. Оба объекта имеют много общих свойств и методов. Однако некоторые свойства и методы уникальны для выделенных областей, а другие — для диапазонов. Значительные различия состоят и в том, что объект Selection соответствует выделению в области окна: тексту, графическому изображению или любому другому объекту, в то время как объекты Range существуют независимо от выделенной области и всегда содержат текст.

Используйте объект Selection в том случае, если ваша процедура зависит от пользователя, например, он должен указать текст, с которым будут проведены определенные действия, или в том случае, если вам необходимо показать пользователю, какой именно текст будет изменен. В других ситуациях намного лучше подходят объекты Range. Они обеспечивают большую скорость выполнения программ и меньше отпугивают пользователя: Word обновляет содержимое экрана при каждом изменении содержимого выделенной области, а при изменении диапазона содержимое экрана не обновляется. Кроме того, изменения диапазонов не отражаются на выделенных областях, созданных пользователем.

Выделение диапазонов и создание диапазонов на основе выделенных областей

Несмотря на их различия, объекты Selection и Range можно создавать один из другого. Эта возможность оказывается чрезвычайно важной, так как многие функции редактирования работают только с диапазонами. В противоположность этому, единственный способ отобразить содержимое диапазона пользователю — выделить его. Используйте следующие простые приемы.

- ✓ Для выделения диапазона используется его метод Select. Например, для объекта RangeR команда имеет вид RangeR. Select.
- ✓ Для получения доступа к диапазону, представляющему то же содержимое, что и выделенная область, используется свойство Range.

Помните: если метод, относящийся к текстовым данным, вызывается для диапазона, а вы хотите применить его к выделенной области, просто включите в свой код инструкцию Selection . Range . *ИмяМетода*.

Повторное определение диапазонов и выделенных областей

Word VBA предлагает целый ряд методов для перемещения и изменения размеров диапазонов и выделенных областей. В настоящем разделе я рассматриваю только самые важные из них; для знакомства с другими методами вам придется обращаться к справочной системе.

Расширение диапазонов и выделенных областей

Метод Expand увеличивает существующий диапазон или выделенную область, добавляя блок текста в их конец. Блок может представлять собой символ, слово, абзац или что-нибудь другое. Вы можете добавить только один заранее определенный блок; кроме того, добавление подобных блоков в начале диапазона или выделенной области не допускается. Для добавления к выделенной области слова, которое будет следовать сразу за ней, используйте такую инструкцию:

```
Selection.Expand(wdWord)
```

Вы можете использовать любую из следующих констант для расширения объекта; wdCharcter, wdWord, wdSentence, wdParagraph, wdSection, wdStory, wdCell, wdColumn, wdRow, wdTable и (только для объектов Selection) wdLine. По умолчанию используется константа wdWord.

Теперь нам следует остановиться на одном щекотливом моменте: объекты Selection (но не диапазоны) также содержат метод Expand. Этот метод включает соответствующее средство Word, которое позволяет расширять выделенную область при перемещении указателя мыши. Каждый раз, когда программа вызывает метод Extend, выделенная область увеличивается на блок текста, который следует сразу за точкой вставки: текущее слово, предложение, абзац, выделенный фрагмент или целый документ. Если вы указали аргумент в виде одного символа, например, Selection.Expand("C"). выделенная область будет расширена вплоть до первого встретившегося указанного символа.

Перемещение диапазона или выделенной области

Word VBA позволяет вам повторно определять начало и конец диапазона или выделенной области. Только имейте в виду, что методы, в имени которых присутствует слово Move, изменяют расположение диапазона или выделенной области: они не перемещают текст, который содержится в указанном объекте.

Метод Move изменяет диапазон или выделенную область, начиная с их сжатия, отмечая их расположение, а в них уже нет никакого текста. Расположение совпадает с началом исходного объекта. После этого метод Move перемещает "сжатый" объект в соответствии с вашими инструкциями. По окончании перемещения вы можете использовать методы Expand и MoveEnd для наполнения объекта текстом.

Приведенный ниже пример перемещает именованный диапазон в документе на два абзаца назад. Обратите внимание на то, что вы используете именованную константу в качестве значения аргумента Unit (список всех допустимых именованных констант приведен в разделе "Расширение диапазонов и выделенных областей" раньше в этой главе). Аргумент Count представляет собой целое положительное число, если вы хотите перемещать объект вперед по документу (т.е. к его концу), или отрицательное, если вы хотите перемещать объект назад по документу. В приведенном ниже примере аргументы не заключены в скобки, поскольку возвращенное методом значение (количество перемещенных элементов) здесь не используется:

```
oTheRange.Move Unit := wdParagraph, Count := -2
```

Методы MoveStart и MoveEnd работают практически так же, как и метод Move, только они изменяют начальную или конечную точку диапазона или выделенной области соответственно. Приведенная ниже инструкция перемещает начало выделенной области на три слова ближе к концу документа:

```
Selection.MoveStart Unit := wdWord, Count := 3
```

Обратите внимание на то, что если вы перемещаете начальную точку объекта в конец, Word сожмет диапазон или выделенную область и переместит их в соответствии с указанными инструкциями.

Еще одна пара методов, StartOf или EndOf, перемещает или расширяет начало или конец диапазона или выделенной области. Метод StartOf перемещает начало объекта обратно к началу текущего блока, в то время как метод EndOf перемещает конец объекта вперед к концу текущего блока.

Вы можете использовать аргумент Extend с любым методом для контроля над действиями Word. Если перемещаемая сторона объекта уже находится с того края, к которому вы пытаетесь ее переместить, ничего не происходит. Используйте константу wdMove для сжимания объекта или константу wdExtend для перемещения только указанной стороны. Ниже приведен соответствующий пример:

```
Selection.StartOf Unit := wdSentence, Extend := wdMove
```

Сжимаем диапазон или выделенную область

Очень часто вам необходимо *сжать* диапазон или выделенную область в точку, которая не содержит никакого текста. С технической точки зрения свернутые диапазон или выделенная область — это такие диапазон или выделенная область, начальная и конечная точки которых совпадают. Сжатие подобных объектов оказывается важным в тех ситуациях, когда вам необходимо вставить поле, таблицу или другой элемент до или после выделенной области или диапазона, обойдясь без замены текста. (Вы *можете* вставить обычный текст, новые абзацы, а также некоторые другие элементы в "несжатые" диапазон или выделенную область).

Используйте метод Collapse для сжатия диапазона или выделенной области. Вы можете сжать объект к его начальной или конечной точке, используя необязательный аргумент Direction. Приведенная ниже инструкция сжимает выделенную область к ее начальной точке:

```
Selection.Collapse
```

А этот пример сжимает выделенную область к ее конечной точке:

```
Selection.Collapse(Direction:=wdCollapseEnd)
```

Если вы сожмете диапазон, который заканчивается знаком абзаца, к его конечной точке (используя константу wdCollapseEnd), Word разместит сжатый диапазон *после* знака абзаца (это означает, что сжатый диапазон будет находиться в следующем абзаце). Если же вы хотите разместить что-то *перед* знаком абзаца исходного диапазона, вы должны сначала переместить диапазон обратно с помощью метода MoveEnd, используя инструкцию, подобную этой:

```
Диапазон.MoveEnd Unit := wdCharacter, Count := -1
```

Удаление, копирование и вставка текста

Удалить весь текст в диапазоне или выделенной области совсем несложно: просто используйте метод Delete соответствующего объекта. Вы можете использовать и метод Cut, если хотите удалить текст и поместить его в буфер обмена. Конечно же, метод Copy копирует текст в буфер обмена, не оказав влияния на текст в диапазоне или выделенной области.

Вы можете вставить текст, раньше помещенный в буфер обмена, в любой диапазон или выделенную область, воспользовавшись методом Paste этого объекта. Если объект назначения еще не сжат, вставленный текст просто заменит исходный текст в объекте, точно так же как это происходит после выполнения команды Вставить в Word.

Хотя использование буфера обмена для передачи текста из одного места в другое кажется вполне привычным, этот метод не всегда оказывается самым эффективным. Намного удобнее использовать свойства Text или FormattedText диапазона или выделенной области. Залайте эти свойства равными диапазону или выделенной области, содержащим текст, который вы хотите передать, и все. Объект назначения должен быть сжат до тех пор, пока передаваемый текст не должен будет заменить существующий текст в объекте.

Приведенный ниже фрагмент кода передает текст из выделенной области в сжатый диапазон, привязанный к закладке (операцию передачи проделывает четвертая строка кода). В новое место расположения попадает только сам текст; любое форматирование при этом теряется:

```
With ActiveDocument.Bookmarks("TheBookmark")
     Set RangeY = _
     ActiveDocument.Range(Start:=.Start, End=.Start)
Fnd With
RangeY.Text = Selection.Text
```

Для передачи вместе с текстом и его форматирования просто замените свойство Text свойством FormattedText.

Вставка нового текста

Простейшим для запоминания приемом добавления текста является задание свойства Text диапазона или выделенной области равным тексту, который вы решили вставить. Это проиллюстрировано на примере, показанном ниже:

```
Range2.Text = "Эй, эй! А ведь дома-то никого нет!"
```

Просто запомните, что использование свойства Text приводит к замене любого существующего в объекте. Во избежание этого (если только вы действительно не хотите заменить существующий текст), *сначала сожмите объект.*

Используйте методы InsertBefore или InsertAfter объектов Range или Selection для вставки текста в определенном месте документа, не затрагивая при этом существующий текст. Эти методы позволяют вставить новый текст непосредственно перед или после указанного объекта соответственно. Word включает вставленный текст в выделенную область или диапазон.

При использовании любого из методов единственным аргументом является текст, который вы хотите вставить. Приведенный ниже фрагмент кода вставляет новый абзац, содержащий слова Dairy Entry в начало выделенной области (обратите внимание на использование такой константы VBA, как vbCr, для добавления знака абзаца). После этого в конец добавляется новый абзац, который начинается с текущей даты. Если вы выделили целый абзац до выполнения этого кода, абзац, содержащий текущую дату, появится после знака абзаца в выделенной области.

```
Dim strInsertText As String
Selection.InsertBefore "Dairy Entry" & vbCr
strInsertText = "Today" & Chr(146) & "s date is"
strInsertText = _
     strInsertText & Format (Now, "Long date") & ". "
Selection.InsertAfter strInsertText & vbCr
```

Этот пример показывает, каким образом вставленный текст может содержать строковые переменные функции VBA, возвращающие строковые значения, а также литералы и константы VBA. Подробные сведения о форматировании и изменении строковых значений с помощью VBA изложены в главе 11.

Простейший способ добавить новый пустой абзац в документ — вставить знак абзаца (представленный константой vbCr) с помощью свойства Text или методов InsertBefore или InsertAfter. Метод Add, применяемый к коллекциям Paragraphs, работает, но не очень удобен в применении. Используйте его в том случае, если вам необходимо разместить новый абзац в диапазоне или выделенной области, а не в их начале или конце.

Форматирование текста

Несколько свойств диапазона или выделенной области — это ваш ключ к изменению внешнего вида текста. Все эти свойства соответствуют командам из меню Формат программы Word и функционируют следующим образом.

Это свойство	Предоставляет доступ к...
Font	Соответствующие свойства каждого аспекта форматирования символов, такие как Name, Size и Bold. По некоторым причинам вы можете получать непосредственный доступ к самым важным свойствам форматирования объектов Range, не обращаясь к свойству Font, но только не при работе с выделенными областями

Это свойство	Предоставляет доступ к...
Paragrap hFormat	Соответствующие свойства каждого аспекта форматирования абзацев, такие как LeftIndent или LineSpacing
Style	Имя стиля символа или абзаца, примененного к диапазону или выделенной области
Borders	Границы вокруг текста
TabStops	Типы И расположения точек табуляции. Вы можете получить доступ к этим свойствам только с помощью объектов Paragraph, а не непосредственно через диапазоны или выделенные области

Поиск и замена текста с помощью VBA в Word

Хотя это звучит и несколько необычно, но Find — это объект Word VBA. Объекты Find принадлежат диапазонам и выделенным областям. Для обнаружения или форматирования текста с помощью объекта Find вам потребуется выполнить следующие действия.

1. **Получите доступ к объекту Find для определенного диапазона или выделенной области. Если вам необходимо просмотреть целый документ, используйте свойство Content объекта Document для получения доступа к соответствующему диапазону, как показано ниже:**

   ```
   ActiveDocument.Content.Find
   ```

2. **Определить свойства объекта** Find **в соответствии с тем, что же вы ищете и как именно вы хотите проводить поиск.**
3. **Вызвать метод Execute объекта** Find. **Соответствующий пример приведен ниже:**

   ```
   With OpenRange.Find
        .ClearFormatting
        .Text = "pogo sticks"
        .Execute
   End With
   ```

Для свойств, значения которых явно вы задать не можете, объект Find выбирает параметры, использованные последними или те, которые в настоящий момент заданы в диалоговом окне Найти и заменить программы Word. Именно по этой причине вам всегда следует включать метод ClearFormatting перед началом нового поиска — он позволяет убрать все ранее определенные для проведения поиска параметры форматирования.

Работа с найденным текстом

Основная работа метода Execute — обнаружение первого экземпляра искомого текста или форматирования в указанном диапазоне или выделенной области. После выполнения этого метода вам прежде всего следует определить, было ли найдено то, что вы ищете. Для подобной проверки используйте свойство Found объекта Find совместно с инструкцией If...Then, как показано на примере следующей заготовки программного кода:

```
If .Found = True Then
     {выполнение определенных действий с найденным текстом}
Else
     {отображение соответствующего сообщения)
End If
```

Если **метол** Execute **нашел необходимый текст, исходный диапазон или выделенная область переопределяются таким образом, чтобы содержать найденный текст. Это очень важный момент, поскольку это означает, что вы можете работать с найденным текстом посредством свойств и методов исходного объекта. В следующем примере, представляющем собой расширенный вариант первого фрагмента кода из настоящего раздела, инструкция** .Parent.Italic = True **обращается к родительскому объекту по отношению к объекту** Find, **т.е. диапазону** OpenRange. **При выполнении этой инструкции диапазон** OpenRange **теперь содержит только найденный фрагмент текста, так как только в нем предусмотрено форматирование курсивом:**

```
With OpenRange.Find
     .ClearFormatting
     .Text = "pogo sticks"
     If .Found = True Then
          .Parent.Italic = True
     Else
          MsgBox "No pogo sticks found."
     End If
End with
```

Замена текста или форматирования

Объект Replacement **принадлежит (а значит, является его свойством) объекту** Find. **При написании кода для операции поиска и замены вам следует задать свойства и выполнить методы объекта** Replacement.

Следующий фрагмент кода заменяет все экземпляры фразы pogo sticks **словом** skateboards. **Выделенная область изменяется при выполнении критерия поиска, поскольку доступ к объекту** Find **осуществляется через объект** Selection:

```
With ActiveDocument.Content.Find
     .ClearFormatting
     .Text = "pogo sticks"
     With .Replacement
          .ClearFormatting
          .Text = "skateboards"
     End With
     .Execute Replace := wdReplaceAll
End With
```

Обратите внимание на то, что метод Execute **может использовать аргумент** Replace, **предназначенный для контроля за тем, будут ли заменены все обнаруженные экземпляры обнаруженного фрагмента текста, или только первый.**

Поиск и замена форматирования

Для поиска текста с определенным форматированием используйте свойства объекта Find, касающиеся форматирования. Они идентичны свойствам, используемым при работе с форматированием диапазона или выделенной области, как я уже отмечал в разделе "Форматирование текста", раньше в настоящей главе. Вы можете использовать те же свойства объекта Replacement, если хотите указать форматирование для замещающего текста.

Для поиска *любого* текста с определенным форматированием задайте соответствующие свойства объекта Find, а также задайте свойство Text равным пустой строке, используя пару кавычек. Для изменения форматирования найденного текста без влияния на сам текст, используйте пустую строку в качестве значения свойства Text объекта Replacement. Приведенный ниже код проводит поиск абзацев, которым в данный момент назначен стиль Drab, после чего назначает им стиль Frilly:

```
With Selection.Find
      .ClearFormatting
      .Style = "Drab"
      .Text = ""
      With .Replacement
           ClearFormatting
           .Style = "Drilly"
           .Text = ""
      End With
      .Execute Replace := wdReplaceAll
      .ClearFormatting
      .Replacement.ClearFormatting
End With
```

Включение двух инструкций для "очистки формата" в вашу процедуру *после* метода Execute — очень неплохая идея. В противном случае, когда пользователь в следующий раз прибегнет к диалоговому окну **Найти и заменить,** ему придется сбрасывать все параметры форматирования вручную.

использование переменных документа

Отличаясь от остальных приложений Office, Word позволяет определять в вашем коде специальные *переменные документа,* которые сохраняются вместе с документом. Переменные документа позволяют сохранять используемые процедурой значения между сеансами редактирования.

Переменные документа создаются и используются как члены коллекции Variables в данном документе. Как и обычные документы, переменные документа характеризуются именами. Приведенная ниже инструкция присваивает значение переменной Henry обычной переменной FriendOfAnais:

```
FriendOfAnais = _
     ActiveDocument.Variable("Henry").Value
```

Для создания новой переменной документа используется метод Add коллекции Variables, как показано ниже:

```
Documents("Document1").Variables.Add _
     Name := "TimeThisMacroHasRun", Value := 0
```

Вы получите сообщение об ошибке, если попытаетесь добавить уже существующую переменную документа, поэтому я рекомендую вам проверить существование соответствующего имени, прежде чем создавать новую переменную. Если это так, вы сможете получить текущее значение переменной; если нет, вы можете создать переменную и назначить ее исходное значение. Этот прием проиллюстрирован следующим примером:

```
For Each DocVar In ActiveDocument.Variables
    If DocVar.Name = "LastCaption" _
        Then DocIndex = DocVar.Index
Next DocVar
If DocIndex = 0 Then
    ActiveDocument.Variables.Add _
        Name := "LastCaption", Value := 1
    CaptionCounter = 1
Else
    CaptionCounter = _
        ActiveDocument.Variables(DocIndex).Value
End If
```

Несмотря на то, что объектные модели других приложений Office не позволяет явно создавать переменные документа, вы можете создавать пользовательские свойства документа для решения определенных задач. Подробные сведения об использовании пользовательских свойств в качестве переменных документа изложены в главе 14.

Глава 16

VBA-программирование в Excel

В этой главе...

- Что такое объектная модель Excel
- Управление ячейками с использованием объектов диапазонов
- Создание собственных функций для использования в формулах рабочих листов
- Использование встроенных функций Excel в VBA-коде
- Работа с событиями Excel

Каждого, кто когда-либо писал формулы рабочих листов, можно считать в некотором роде программистом; поэтому не следует относиться к VBA, как к чему-то запредельному. Скорее всего, после некоторой работы с редактором Visual Basic вы обнаружите, что писать VBA-код, на самом деле, удобнее, чем составлять формулы непосредственно в рабочем листе; VBA предоставляет больший простор для деятельности в окне редактирования кода и позволяет вносить комментарии, что бывает очень полезным.

Кроме того, VBA может служить мощным инструментом для создания настраиваемых приложений, базирующихся на таблицах, способных выполнять вычисления и форматирования, которые без них просто не реачьны. Но прежде чем стать профессионалом в использовании кодов Excel, чему способствует представленный ниже материал, следует ознакомиться с основами VBA, которые даются в первых трех разделах.

Знакомство с объектной моделью Excel

Прежде чем приступить к написанию VBA-кода для Excel, необходимо составить представление об иерархии объектов Excel и о том. как определять их в написанном коде. В самом верху иерархии находится объект `Application` (Приложение), отвечающий за функционирование всей программы Excel.

Для увеличения скорости работы многих VBA-программ следует отключить обновление экрана. По умолчанию Excel отображает каждое изменение в рабочей книге, сделанное в ходе выполнения VBA-кода. Это приводит к существенному снижению быстродействия. Для отключения такого режима работы Excel необходимо воспользоваться функцией `ScreenUpdating` (Обновление экрана) объекта `Application`:

```
Application.ScreenUpdating = False
```

Не забудьте в конце процедуры восстановить значение `True` для свойства `ScreenUpdating`, в противном случае пользователь не сможет увидеть на экране полученный результат.

При использовании свойства `ScreenUpdating` необходимо явно указывать имя объекта `Application`. Однако в большинстве случаев свойства объекта `Application` можно использовать напрямую. Например, свойство `ActiveSheet` объекта `Application` отно-

сится к рабочему листу или диаграмме, активной в текущий момент (безусловно, в книге, активной сейчас). Для обращения в коде к такому листу вместо записи Application.ActiveSheet можно использовать просто ActiveSheet. Коллекция Workbooks объекта Application содержит все рабочие книги, открытые в данный момент. Для идентификации определенной рабочей книги используйте ее имя, заключенное в кавычки. Вот пример активизации отдельной рабочей книги:

```
Workbooks("Сводная статистика продаж.xls").Activate
```

Конечно же, рабочие листы также являются полноправными объектами. Они принадлежат коллекции Worksheets, и их идентификация должна проводиться таким же образом. Выражение Worksheets("Лист3") указывает на лист с именем Лист3. Аналогично, отдельный объект Chart для диаграммы, находящейся на отдельном листе, является членом коллекции Charts. Обращение к ней выполняется с помощью выражения вроде Charts("Parts Chart"). Приемы работы с диаграммами, внедренными в рабочий лист, несколько отличаются, и приемы написания кода для них являются объектом рассмотрения данной книги.

Отображение рабочих книг можно изменить с помощью объектов Window (Окно). Объекты Window, которые иногда используются и для изменения содержимого рабочей книги, являются членами коллекции Windows объекта Application. Ссылка на окно осуществляется указанием в качестве индекса в коллекции Windows имени файла, содержащего рабочую книгу:

```
Windows("Кривая продаж игрушек.xls")
```

Если для одной рабочей книги открыто несколько окон, необходимо после имени рабочей книги через двоеточие указать номер окна, например: Windows("Кривая продаж игрушек.xls:2").

Использование в коде объектов Range для работы с ячейками

Весьма неожиданно, что в Excel нет объекта Cell (Ячейка). Поэтому при написании кода следует руководствоваться следующим подходом: для указания ячейки в VBA-коде используется объект Range. В Excel объект Range может заключать в себе одну или больше ячеек и даже несколько не непрерывных областей листа.

Объект Range в Excel во многом подобен аналогичным объектам в Word, но вместе с тем имеет и существенные отличия. Как и в Word, VBA-программа может ссылаться на любое необходимое число объектов Range. Как в Excel, так и Word, действие программы не ограничивается видимым выделением пользователя, и для воздействия на какую-либо область ее выделять не требуется.

Определение объекта Range

В Excel имеется несколько возможных методов идентификации диапазона из одной или нескольких ячеек, на который должен воздействовать написанный код. Для достижения данной цели используются следующие.

Стандартная ссылка на ячейку. Так называемый A1-стиль ссылки на ячейку является, пожалуй, самым простым способом работы с объектами Range. Для определения диапазона необходимо ссылку заключить в кавычки и скобки после ключевого слова Range, как показано в следующем примере:

```
ActiveSheet.Range("B3")
Worksheets("Sheet2").Range("M5:S20")
```

Именованные диапазоны. Если рабочий лист содержит именованные диапазоны, VBA-объекты Range могут опираться на них, как показано в следующем примере:

```
Worksheets("Финансовый отчет").Range("Выплата процентов")
```

Для присвоения диапазону имени непосредственно в самом коде необходимо употребить следующее выражение с использованием функцией свойства Name (Имя):

```
Range("A3:B4").Name = "Прайс-лист"
```

Сокращенная запись. Вследствие частого использования объектов Range, Excel позволяет упускать ключевое слово Range при определении диапазона при записи в A1 -стиле или при записи с использованием имени. Для реализации такого приема ссылку на ячейку или имя диапазона следует заключить в квадратные скобки, как показано в приведенном ниже примере:

```
ActiveSheet["A1:Z26"]
["Квартальный отчет"]
```

Свойство Cells объекта Worksheets. Данная техника крайне необходима профессионалам, так как позволяет определять диапазон не путем указания фиксированных адресов ячейки, а на основе переменных. Основная идея состоит в составлении в цифровом виде списка координат строк и столбцов диапазона. Читайте об этом в разделе "Использование свойства Cells для определения диапазона" дальше в данной главе.

Свойство Selection (Выделение). Когда требуется воздействие кода на диапазон, соответствующий выделению пользователя, используется свойство Selection. Читайте об этом в разделе "Работа с выделениями" дальше в данной главе.

Свойство ActiveCell (Активная ячейка). Свойство ActiveCell используется для доступа к диапазону, представляющему активную ячейку данного окна. При использовании без спецификатора объекта (что эквивалентно использованию объекта Application), свойство ActiveCell ссылается на активное окно:

```
ValueStorageBin = ActiveCell.Value
```

Свойства Rows (Строки) или Columns (Столбцы) объекта Worksheet. Доступ к диапазону, включающему весь столбец или строку, осуществляется с помощью свойств рабочего листа Rows и Columns с использованием номера указываемого столбца или строки (нельзя адресовать столбец через его буквенное обозначение). В следующем примере определяется диапазон, включающий столбец E, т.е. пятый столбец:

```
Workbooks("IOU.xls").Worksheets("Sheet Shootout").Column(5)
```

Определенные пользователем ссылки на объекты. Поскольку диапазон является объектом, можно установить именованную объектную ссылку на него, после чего доступ к диапазону осуществляется с помощью указания имени ссылки. Данную технику использовать проще и быстрее, чем многократное указание оригинального диапазона. После установки объектной ссылки RanGer, как показано в последующем примере, можно использовать его свойства в таких выражениях, как RanGer . Value:

```
Dim RanGer As Range
Set RanGer = Worksheets("Лист1").Range("B12:H13")
```

Использование свойств Cells для определения диапазона

При использовании без координат свойство Cells объекта Worksheets указывает на диапазон, включающий *все* ячейки данного рабочего листа. По аналогии, свойства Cells

объекта Application (Application.Cells) ссылаются на все ячейки листа, активного в данный момент (свойство Cells может использоваться само по себе, без указания в явном виде объекта Application).

Если необходимо остановиться на более локализированном диапазоне, для свойства Cells требуется числовое указание координат строки и столбца (буквенное указание столбца не допускается). В следующем примере указан диапазон, заключающий ячейку E3:

```
Worksheets("Старые новости").Cells(3,5)
```

Весьма непривычно то, что сначала указывается координата строки, а затем столбца, в противоположность записи в A1-стиле. В предшествующем примере второе значение в ссылке на ячейку указывает на столбец E, т.е. пятый столбец. Вследствие того, что данная система трудна для понимания, работать с ней следует лишь в случае особой необходимости. Поскольку обе координаты являются числами, можно очень просто указывать их через переменные. Переменные координаты позволяют программе (при работе) принимать решение о том, где находится требуемый диапазон на основе введенных пользователем данных, результатов вычислений и т.д. В следующем примере строка выбирается в зависимости от текущего года и месяца:

```
intMonth = Month(Now())
aGoal = Worksheets("Monthly Projections").Cells(intMonth, 8)
```

Способом чуть похитрее можно указать диапазон, покрывающий больше одной ячейки:

```
Range(Cells(3,5), Cells(4,6))
```

В приведенном здесь примере указывается диапазон ячеек 2x2 на активном рабочем листе, включающий ячейки E3 (строка 3, столбец 5) в верхнем левом и ячейку F4 в левом нижнем углах.

Для определения диапазона на неактивном листе необходимо использовать совместно свойства Range и Cellsдля требуемого листа. Следующее выражение выполняет данную работу, правда, имеет достаточно громоздкий вид:

```
Worksheets("Лист2"). _ Range(Worksheets("Лист2").Cells(3,5), _
Worksheets("Лист2").Cells(4,6))
```

Оператор With позволяет избежать излишних ссылок на рабочий лист. В следующем примере к тексту всего диапазона применяется полужирное форматирование:

```
With Worksheets("Лист2")
     .Range(.Cells(3,5), .Cells(4,6)).Font.Bold = True
End With
```

Обратите внимание, что точки перед каждым использованием свойства Cells необходимы для связи любой ссылки с требуемым рабочим листом. В случае отсутствия точек каждое свойство Cells будет относится к активному листу, что приведет к возникновению конфликта со ссылкой свойства Range.

Выполнение совместных действий с ячейками

Используя свойства диапазона, можно одним действием изменять характеристики целого диапазона. Следующая строка изменяет размер шрифта текста во всех ячейках диапазона:

```
Worksheets("Лист1").Range("B12:H13").Font.Size = 14
```

Безусловно, можно и даже предпочтительно использовать структуру With, когда необходимо работать с несколькими свойствами или методами целого диапазона, как показано здесь:

```
With someRange ' определенная ранее объектная ссылка
     .Value = 20 ' значение всех ячеек устанавливается равным 20
     .Font.Name = "Garamond"
     .Font.Italic = True
     .Locked = True
     someRange = .Name ' сохраняется имя диапазона
End With
```

Работа с отдельными ячейками диапазона

Хотя можно с помощью одного оператора назначить одно значение всем ячейкам диапазона, как показано в предыдущем примере, в Excel нет метода, позволяющего с помощью единственного действия изменять имеющиеся значения многоячеечного диапазона. Оператор вроде someRange.Value = someRange.Value + 10 не работает. Вместо этого необходимо осуществить циклический перебор всех ячеек диапазона с помощью цикла For Each. . , Next. При использовании данной техники не требуется знание количества ячеек, входящих в диапазон. Вот пример работоспособного кода:

```
For Each aCell In Selection
     aCell.Value = aCell.Value + 10
Next
```

Часто перед тем, как принять решение о выполнении действия над отдельной ячейкой или о вообще о выборе действия, требуется проверка содержимого ячейки. Основываясь на текущем значении ячейки, код может принять решение о форматировании ячейки, изменении ее значения или же использовании данного значения для выполнения других вычислений. В таком случае также используется цикл For Each. . . Next:

```
For Each aCell In Worksheets("Лист2").Range("A5:B10")
     If IsNumeric(aCell) Then
          Select Case aCell
               Case 5 to 10
                    aCell.Font.Underline = xlUnderlineStyleSingle
               Case 10 to 20
                    aCell.Font.Italic = True
               Case Is > 20
                    aCell.Font.Bold = True
          End Select}
     End If
Next
```

Работа с выделениями

Прежде чем изменить значение или формат ячейки или нескольких ячеек, пользователь должен их выделить. Однако в VBA выделение ячеек не требуется, так как для идентификации ячеек, над которыми должны проводиться определенные действия, можно использовать объекты Range. Но в VBA есть инструменты, связанные с выделением; они используются для реализации двух возможностей — код способен определить, какие ячейки выделены пользователем, и код может показать пользователю место на рабочем листе, где происходит что-то важное.

Определение текущего выделения

Во многих случаях пользовательский код должен воздействовать на выбранные ячейки, как это делают встроенные команды Excel. Для доступа к выделенному пользовате-

лем диапазону используется свойство Selection (Выделение) объекта Application или Window. Свойство Selection объекта Application возвращает диапазон, выделенный на рабочем листе, активном в данный момент. Следующие два оператора идентичны:

```
Application.Selection.Value = 20
Selection.Value =20
```

Для того чтобы удостовериться в том, что ссылка всегда относится к определенному окну, независимо от того, какое окно активно в текущий момент, необходимо использовать свойство окна Selection. В следующем примере продемонстрирована данная техника; также здесь показано, как установить объектную ссылку на диапазон, представляющий текущее выделение для повторного использования того же диапазона в дальнейшем:

```
Dim SelRange As Range
Set SelRange = Windows("Инвентаризация игрушек.xls").Selection
With SelRange
      .CheckSpelling
      .AutoFit
      .Copy
End With
```

Выделение диапазона

Когда VBA-программа вносит изменения в рабочий лист и необходимо, чтобы пользователь их заметил, следует воспользоваться методом Select (Выделить) для перемещения выделения на нужный диапазон, диаграмму или что-либо другое. Метод Select можно применять фактически к любому существующему в Excel объекту, в том числе к объектам Chart (Диаграмма) и всем их компонентам (каждая часть диаграммы является отдельным VBA-объектом). к объектам Shape (Форма) и, конечно же, к объектам Range.

Для выделения диапазона необходимо сначала активизировать рабочий лист, на котором диапазон находится, а затем использовать метод Select для объекта Range, как показано в данном примере:

```
With Worksheets("Статистика любви")
      .Activate
      .Range("Разбитые сердца").Select
End With
```

Кстати, метод Select для объектов Worksheet, очевидно, не выполняет ничего, кроме активизации указанного рабочего листа, и не изменяет в нем существующее выделение. Другими словами, он эквивалентен методу рабочего листа Activate. Подобным образом методы Activate и Select можно использовать для активизации листа диаграммы, но ни один из методов в действительности не выделяет диаграмму. Вот пример;

```
Charts("Места жительства клиентов").Select
```

Для того чтобы выделить компонент диаграммы или внедренную диаграмму, используйте метод Select для интересующего вас объекта.

Активизация определенной ячейки

Чтобы сделать ячейку активной для ввода, к данной ячейке используется метод Activate (Активизировать) объекта Range. Если активизированная ячейка находится в пределах текущего выделения, весь диапазон остается выделенным. Именно таким образом работает следующий пример:

```
Worksheets("НичегоНеДелайДоМоегоУказания").Activate
Range("A1:E7").Select
Range("C4").Activate
```

Если активизированная ячейка находится за пределами выделения, оно перемещается на активизированную ячейку.

Определение типа выделения

Код для работы с выделенными ячейками, вероятно, приведет к возникновению ошибки, если его применить к диаграмме. Вот почему, прежде чем выполнять какие-либо операции с выделением, обычно следует убедиться, что выделение содержит тот тип объектов, который ожидается в коде. VBA-функция TypeName возвращает строку, содержащую тип объекта выделения. С помощью структур If. . .Then или Case. . .Select можно определить, какое из возможных действий проводить над выделенным объектом.

В приведенном ниже примере выражение TypeName (Selection) в операторе Select (вторая строка кода) возвращает строку, содержащую тип объекта текущего выделения. Данная строка затем проверяется рядом операторов Case, является ли выделение диапазоном (в таком случае его значение устанавливается равным 2001), областью диаграммы (в таком случае устанавливается красный цвет выделения), когда же выделение не сделано, пользователь видит сообщение об отсутствии выделения. Также предусмотрен вывод соответствующего сообщения для случая, когда выделение не принадлежит ни одному из перечисленных типов:

```
With Selection
Select Case TypeName(Selection)
     Case "Range"
          .Value = 2001
     Case "ChartArea"
          .Interior.ColorIndex = 3 ' 3 =bright red
     Case "Nothing"
          MsgBox "Ничего на выделено"
     Case Else
          MsgBox "Невозможно определить тип выделения!"
End Select
End With
```

Программирование пользовательских функций

Даже когда кажется, что встроенные инструменты анализа данных Excel способны выполнить все необходимые действия, работа с VBA может оказаться более удобной. Использование VBA позволяет создавать пользовательские функции для рабочих листов, существенно превосходящие формулы, которые могут записываться непосредственно в ячейке.

Пользовательские функции позволяют проводить вычисления и другие операции, выполнение которых с помощью формул, основанных на встроенных функциях, просто невозможно. Даже когда написанная формула дает такой же результат, пользовательская функция имеет существенные преимущества, например в большей простоте написания, тестирования, понимания и т.д. Вместо тесной панели формул в распоряжение предоставляется целое окно редактирования кода, где сложную логическую конструкцию мож-

но разбить на понятные линии. Еще одно важное преимущество состоит в возможности вставки комментариев (и это следует максимально использовать) непосредственно рядом с кодом, к которому они относятся.

Написание пользовательских функций рабочего листа

Пользовательские функции Excel— это просто обычные процедуры VBA-функций. Детально с процедурах функций и их синтаксисе можно узнать из главы 6. Если сказать коротко, процедура функции начинается с декларации ее имени и заканчивается оператором `End Function`. Иногда внутри может потребоваться оператор, присваивающий значение имени функции, данное значение как раз и возвращает функция. В простейшем примере, приведенном здесь, именно это и демонстрируется:

```
Function MemoryAvailable()
     MemoryAvailable = Application.MemoryFree
End Function
```

Данная функция просто возвращает количество памяти в байтах, доступных в текущее время для Excel. Заметьте, что, поскольку функция получает данные о количестве доступной памяти из системы, она не имеет аргументов. Ниже приведен пример немного более сложной функции, принимающей аргументы:

```
Function CheckForValue(aRange,Value)
For Each objCell In aRange
     CheckForValue = False 'по умолчанию возвращается значение
False
     If objCell.Value = Value Then
          CheckForValue = True
          Exit For
     End If
Next objCell
EndFunction
```

Данная функция проверяет диапазон ячеек на наличие определенного значения. Если значение есть где-либо в диапазоне, функция возвращает значение `True`, в противном случае возвращается значение `False`. Перед инициированием функции необходимо обеспечить ее двумя аргументами — диапазоном и искомым значением.

Запуск пользовательских функций

Один из способов запустить функцию, — это, конечно же, использовать стандартную процедуру запуска в VBA, т.е. вызвать ее внутри процедуры Sub. Детально данная техника описана в главе 6.

Для того чтобы вставить в рабочий лист возвращаемое функцией значение, следует использовать функцию таким же образом, как используется любая из встроенных функций: внести имя функции в ячейку после знака равенства. После имени печатаются круглые скобки с любым аргументом внутри. Скобки необходимы даже в случае, когда аргументы отсутствуют, как в следующем примере:

```
=MemoryAvailable()
```

Так же как и встроенные, пользовательские функции могут быть частью более сложной формулы ячейки, как в приведенном примере:

```
=MemoryAvailable() & ""сейчас доступно байт"
=IF(CheckForValue(B8:B18,C8)),"Значение найдено","Значение не
найдено")
```

Использование Панели формул с пользовательскими функциями

Действительно замечательно то, что **Панель формул** распознает пользовательские функции. Если вы не помните, какие аргументы требуются для функции и принимает ли функция вообще какие-либо аргументы, не стоит беспокоится, **Панель формул** показывает все, что необходимо для функции

Можно добавить к функции описание, которое будет появляться в диалоговом окне Вставка функции при выборе функции. Для создания описания выберите команду Сервис⇨Макрос⇨Макросы и затем в поле **Имя макроса** введите имя функции (функции не отображаются в списке макросов). Теперь щелкните на кнопке **Параметры** для ввода описания функции в соответствующее поле.

Доступ к пользовательским функциям из других рабочих книг

Если нужная функция хранится в текущей книге, достаточно просто написать ее имя в ячейке формулы. Для того чтобы использовать функцию, хранящуюся в другой *открытой* книге, имени функции должно предшествовать имя рабочей книги и восклицательный знак, как показано в следующем примере:

```
=Примитивныефункции.xls!УмнаяФункция(C4:D6,M9)
```

Для доступа к функциям, хранящимся в книгах, не открытых в текущий момент или закрытых при возникновении доступа к ним в дальнейшем, необходимо настроить на них ссылку VBA. Выберите команду Tools⇨References. Если нужная рабочая книга не представлена в списке диалогового окна, щелкните на кнопке **Browse** для ее локализации и добавления в список. Если флажок рядом с книгой не установлен, сделайте это для активизации ссылки. Теперь любую из содержащихся в книге функций можно использовать, просто указав ее имя, не указывая имени книги.

Тестирование пользовательских функций

Если при выполнении пользовательской функции, помещенной в формулу рабочего листа, происходит ошибка, обычное сообщение об ошибке VBA-функции не выводится. Все что можно увидеть вместо этого, — это неясное сообщение об ошибке вроде `#ЗНАЧ!` в ячейке, содержащей формулу. При построении пользовательской функции можно воспользоваться следующими приемами, которые помогут в тестировании и отладке функции.

Прежде чем поместить функцию в рабочий лист, протестируйте ее, вызвав через процедуру Sub. Такой способ вызова позволяет получить сообщение об ошибке VBA и доступ к инструментарию отладки. Если функция требует ссылки на ячейки, необходимо при вызове функции использовать объект Range, как показано в данном примере:

```
Sub FxTester()
    ReturnVal = CheckForValue(Range("B8:B13"),Range("C8"))
    MsgBox ReturnVal
End Sub
```

В коде функции, записанной в формуле рабочего листа, в редакторе Visual Basic создайте точку разрыва. Функция будет запускаться при каждом пересчете Excel рабочего листа. Как только VBA достигнет строки, содержащей точку разрыва, вы автоматически попадете в редактор, готовый к выполнению отладки.

Профессиональное написание функций

Не стоит отказываться от возможности написания функции таким образом, чтобы выдаваемый се результат был именно таким, как вам того хотелось бы. Зачем поручать формуле выполнять то, что можно сделать в самой функции? Приведенная модификация функции CheckForValue (она приводилась в разделе "Написание пользовательских функций рабочего листа") вместо непонятных значений True и False возвращает строку поясняющего текста:

```
Function CheckForValue2(aRange, Value)
For Each objCell In aRange
     CheckForValue2 = "искомое значение" 6 Value & _ "не найдено"
     I;: objCell.Value = value Then
          CheckForValue2 = "искомое значение" & Value & _ "находится
          в ячейке" & objCell.Address
          Exit For
     End If
Next objCell
End Function
```

Если модифицированной функции удается найти искомое значение где-либо внутри диапазона, она возвращает строчку вроде Искомое значение 3, 57 находится Б ячейке F83; в противном случае возвращается строчка Искомое значение 3,57 не найдено.

Использование в scoqe встроенных функций

Независимо от того, пишете ли вы собственную функцию или процедуру Sub, не стесняйтесь призывать на помощь широчайшие возможности встроенных функций Excel по анализу данных и проведению вычислений. Использовать их в своем собсгвенном коде очень просто: необходимо вызвать их как метод объекта WorksheetFunctions. Предположим, что нужно написать процедуру, проводящую вычисления на основе среднего значения чисел определенного диапазона. Дня получения среднего значения можно использовать код, подобный следующему:

```
OnAverage = WorksheetFunction.Average(Range("B8:B13"))
```

Некоторые из встроенных функций Excel запрещены в VBA. Это касается функций, дублирующих встроенные VBA-функции, о которых шла речь в главе 11.

Программирование событий Excel

Реакция на вносимые пользователем изменения стала неотъемлемой частью функционирования электронных таблиц, поэтому написание кода для событий в Excel часто играет более значимую роль, нежели в остальных приложениях пакета Office. Изменение значения лишь одной ячейки, использующейся в расчетах формулы или диаграммы, может привести к существенным последствиям для всей рабочей книги. Возможность перехвата событий, управляющих данным процессом, для улучшения и расширения функциональности встроенной системы ответов Excel, целиком находится во власти VBA-программиста.

Выбор правильного объекта

Перед написанием кода обязательно нужно решить, какой объект должен отвечать на событие. В Excel способны распознавать события четыре объекта: диаграммы, отдельные рабочие листы, рабочие книги и приложение Excel в целом. Если требуется написание кода, отвечающего на событие, связанное с диаграммой, такой код подходит к процедуре события диаграммы. Однако для событий, являющихся ответом на изменения **в** рабочем листе, программист получает несколько вариантов выбора.

Хотя некоторые события Excel распознают лишь определенные объекты, большинство из них образует иерархию от объекта Worksheet до объектов Workbook и Application. Например, изменения, внесенные в рабочий лист, инициируют событие Change (Изменение) объекта Worksheet, который в свою очередь инициирует событие SheetChange (Изменение листа) для объектов Workbook **и** Application. Принимая во внимание сказанное выше, следует определить, подходит ли код для процедуры события рабочего листа, рабочей книги или приложения. Задача выбора не составит много труда.

- ✓ Если код должен выполняться только при ответе на изменения в единственном рабочем листе, он подходит процедуре события данного рабочего листа.
- ✓ Если процедура должна выполняться при внесении изменений в любом рабочем листе определенной книги, код следует вставить в процедуру события рабочей книги.
- ✓ Если же код должен активизироваться в любой из открытых рабочих книг, он безусловно подходит процедуре события приложения.

Испозование процедур событий

Техника написания процедуры события для любого объекта Excel в своей основе не имеет никаких принципиальных отличий по сравнению с техникой написания кода события для формы или элемента управления VBA. В главе 10 приведено детальное описание методики, но основные моменты можно представить следующим образом.

1. **Откройте окно редактирования кода для объекта.**
2. **Выберите объект, выбрав его имя из раскрывающегося списка Объект в левом верхнем углу окна.**
3. **Выберите из раскрывающегося списка Процедура, в правом верхнем углу окна, событие, для которого должен быть написан код.**
4. **При этом в окне появляется каркас выбранной процедуры сообщения.**

Кроме написания самого кода, единственным сложным моментом может быть вынос окна редактирования кода на первое место (п. 1 в приведенном выше списке). Для объектов Worksheet, Workbook и Chart, занимающих отдельный лист, никаких особенностей нет — необходимо просто выбрать объект в окне обозревателя и щелкнуть на кнопке View Code. Объекты представляются в алфавитном порядке, а их имена можно изменять в окне Options.

В случае диаграмм, внедренных в рабочий лист, а также объектов приложения Excel ситуация посложнее. Для того чтобы сделать такие объекты доступными из окна, требуется написание специальных модулей классов. Объем данной книги не позволяет детально рассмотреть данный вопрос, однако информацию, касающуюся данной темы, можно получить из справочной системы.

Реагирование на изменения в рабочем листе

Если необходимо, чтобы код реагировал на действия, производимые пользователем с рабочим листом, потребуются такие инструменты, как свойства Change, Calculate и SelectionChange (для объектов Worksheet) и соответствующие события SheetChange, SheetCalculate и SheetSelectionChange (для объектов Workbook и Application). Для запуска пользовательских процедур в то время, когда сами рабочие листы или диаграммы активизированы или не активизированы, используются события Activate и Deactivate.

События Change и SheetChange

События Change и SheetChange инициируются каждый раз, когда значение любой ячейки или нескольких ячеек изменяется в результате действий пользователя или обновления ссылки. Однако изменения в вычисляемых значениях не приводят к инициированию события. Соответствующие процедуры событий позволяют наметить ячейку, значение которой было изменено. В следующем примере проверяются измененные значения внутри диапазона, называемого Target, на предмет их попадания в определенные пределы; такие значения выделяются с помощью шрифта с большим кеглем, полужирным начертанием и зеленым цветом.

```
Private Sub Worksheet_Change(ByVal Target As Range)
    For Each oCell In Target
        If oCell > 4 And oCell < 11 Then
            With oCell.Font
                .Bold = True
                .Size = 16
                .Color = RGB(0, 255, 0) 'зеленый
            End With
        End If
    Next oCell
End Sub
```

Обратите внимание, что Target может быть не одной ячейкой, а диапазоном, благодаря тому, что операции заливки, удаления и вставки могут применяться одновременно к нескольким ячейкам. Для этого процедура сообщения Change должна включать в себя структуру For Each ... Next; в таком варианте процедура применима как отдельной ячейке, так и к нескольким ячейкам, что и демонстрирует приведенный выше пример.

В действительности, события Change и SheetChange могут активизироваться даже тогда, когда значение не изменялось. Данные события готовы произойти, когда пользователь начинает редактирование ячейки (после щелчка в панели формул или нажатия клавиши <F2>), даже если сразу после этого он прекратил редактирование, не внеся никаких изменений (нажав клавишу <Enter>, щелкнув на кнопке Ввод или на рабочем листе). Событие не активизируется, если пользователь прекращает редактирование нажатием клавиши <Esc> или щелчком на кнопке Отмена.

События Calculate и SheetCalculate

Событие Calculate распознаваемое как объектом Worksheet, так и Chart, происходит при каждом обновлении программой Excel рабочего листа или диаграммы.

Событие SheetCalculate для объектов Workbook и Application происходит в ответ. Если средство автоматического вычисления включено, данное событие активизируется, как только изменяется значение любой ячейки, т.е. происходит в тандеме с событием Change. Когда средство пересчета включено в ручном режиме, событие Calculate происходит лишь тогда, когда пользователь инициализирует пересчет нажатием клавиши <F9>.

Процедуры событий для событий Calculate и SheetCalculate используются для изменения рабочего листа в соответствии с результатами вычислений. Например, если известно, что пересчет может изменить элементы упорядоченного списка, целесообразно использовать процедуру события Worksheet_Calculate для упорядочения списка после проведения вычислений. Поскольку данные процедуры не сообщают, какая из ячеек изменялась в результате вычислений, необходимо поместить в код адреса ячеек, которые требуется изменить.

События SelectionChange и SheetSelectionChange

При каждом перемещении активной ячейки, а также при расширении или сжатии выделения Excel вызывает событие SelectionChange для worksheet. Параллельно происходит событие SheetSelectionChange для объектов Workbook и Application. Процедуры событий для данных событий можно использовать для обратной связи с текущим выделением. В приведенном ниже примере событие SelectionChange используется для отображения в левой верхней ячейке текущего листа адреса активной ячейки, а также для помещения имени рабочего листа и адреса выделения в строку состояния. Обратите внимание на то, как аргумент Sh позволяет идентифицировать и вести работу с текущим листом:

```
Private Sub Workbook_SheetSelectionChange(ByVal Sh As Object, _ ByVal Target As Excel.Range)
    Sh.Range("A1") = ActiveCell.Address
    Application.StatusBar = Sh.Name & ":" & Target.Address
End Sub
```

Несколько усложнив код, можно добиться того, чтобы вместо простого выделения вызывался какой-нибудь ответ: например, чтобы в случае попадания определенной ячейки или диапазона в новое выделение отображалось пользовательское диалоговое окно:

```
Private Sub Worksheet_SelectionChange(ByVal Target As Excel.Range)
    If Target.Address = "$3$2" Then
        MsgBox "Вы нашли нужную ячейку!"
    End If
End Sub
```

Программирование динамических диаграмм

Поскольку объекты диаграмм Excel способны распознавать большое число событий, многие из которых связаны с мышкой, их можно считать большим элементом управления ActiveX, помещенным в рабочей книге. Диаграммы распознают события Activate, Deactivate и Calculate. Вот еще некоторые события, для которых возможно написание кодов.

- ✓ DragOver и DragPlot. Происходят, когда ячейки перемешаются над внедренной диаграммой или опускаются на нее соответственно.

- ✓ `MouseDown`, `MouseUp` и `MouseMove`. Происходят в ответ на действия мыши.
- ✓ `Select`. Происходит при выделении какой-либо части диаграммы. Написанный код может определять тип и характеристики выделенного элемента и проводить соответствующие действия.
- ✓ `SeriesChange`. Происходит, когда пользователь изменяет значение в диаграмме через саму диаграмму (а не изменяя значения, содержащиеся в листе).

Глава 17

Программирование баз данных

В этой главе...

- Знакомство с терминологией
- Написание кода баз данных с помощью объектов данных ActiveX
- Программирование с помощью DAO
- Ускорение с помощью SQL

Несмотря на то, что Access — официальное приложение для работы с базами данных, входящее в состав Office, вы не ограничены только этим приложением при создании пользовательских баз данных с помощью VBA. Действительно, *любое* приложение, поддерживающее VBA, — от Word до CorelDraw, — позволяет использовать сведения, которые хранятся в базах данных на вашем компьютере, сервере локальной сети или любом другом компьютере в Internet. В настоящей главе мы остановимся на нескольких базовых приемах программирования на VBA, позволяющих управлять данными прямо из ваших собственных VBA-программ.

Программирование баз данных на VBA: основные термины

Прежде чем вы сможете создавать собственный программный код для работы с базами данных, вам следует ознакомиться с основной терминологией. Этому и посвящен материал настоящего раздела.

О ядре баз данных

Программное обеспечение, которое отвечает за выполнение основных действий по извлечению информации, которая содержится в одном или нескольких файлах базы данных, называется *ядром базы данных*. Ядро базы данных часто называют просто базой данных, хотя согласитесь, что называть программное обеспечение, управляющее информацией в базе данных, и саму базу данных одним термином несколько нелогично.

Хотя эта программа и предназначена для выполнения различных функций с базами данных, Access нельзя назвать ядром базы данных. Access — это "клиент" (front end), который показывает вам, какие именно команды и данные доступны, а также воспринимает ваши инструкции по отношению к информации, такие как ее изменение, добавление или удаление. Все эти инструкции передаются серверной части (back end) СУБД.

Стандартным ядром баз данных для Access является Microsoft Jet. Стандартные файлы баз данных Access (вы всегда сможете определить их по расширению .mdb) на самом деле яв-

ляются файлами Jet. Работая через Jet, другие инструменты разработки, такие как Visual Basic, могут получить из файлов баз данных все необходимые сведения.

Jet предназначено для обслуживания индивидуальных пользователей и небольших рабочих групп. Для решения более серьезных задач, уровня предприятия, например, предназначены другие ядра баз данных. Сюда относятся еще один программный продукт компании *Microsoft* — SQL Server, а также предложения от других ведущих поставщиков, таких как *Oracle* и *Informix*. Версия SQL Server для разработчика, которая называется SQL Server 2000 Desktop Engine, поставляется в составе Access 2002. Она позволяет вам разрабатывать базирующиеся на технологиях SQL Server приложениях баз данных прямо на своем компьютере, а затем передавать их "настоящей" версии SQL Server, работающей в сети.

В любом случае, всегда можно запутаться, когда дело касается программного обеспечения клиентной и серверной частей СУБД. Работая с Access или любым другим VBA-приложением, вы можете взаимодействовать с различными ядрами баз данных, руководствуясь прежде всего вашими конкретными потребностями. Access предоставляет больше возможностей программисту баз данных, но вы можете использовать Word, CorelDraw или любое другое поддерживающее VBA приложение, и получить при этом не менее превосходные результаты.

SQL и VBA

SQL (Structured Query Language — Язык структурированных запросов) — это общепринятый стандарт организации запросов баз данных (запрос — это команда, которая получает или изменяет информацию, которая содержится в базе данных). Любая система управления данных, будь то Jet или Oracle, поддерживает язык SQL. SQL — это полноправный язык программирования, однако лучше всего он подходит именно для управления базами данных. Вы создаете инструкции SQL для выбора, изменения или удаления определенных наборов записей.

А как же SQL и VBA дополняют друг друга? VBA позволяет передавать ядру базы данных инструкции SQL для их дальнейшей обработки. После обработки инструкций и извлечения необходимого набора записей ядром базы данных вам снова потребуется VBA для отображения и форматирования результатов: SQL при этом не используется.

Все об объектах баз данных

Сам по себе язык VBA не предоставляет никаких средств для доступа к базам данных и манипулирования содержащейся в них информацией. Однако после объединения с *библиотекой объектов* баз данных VBA тотчас превращается в инструмент для программирования баз данных с очень широкими возможностями.

Основная задача, выполняемая библиотекой объектов базы данных, — представление базы данных и всех ее компонентов (таблицы, запросы, отчеты и т.д.) в объекты, по своему проявлению мало отличающиеся от других объектов, используемых в VBA. Благодаря этому вы получаете возможность манипулировать ими, используя их свойства, методы и события (подробности в главе 12). Поскольку база данных представляется в виде набора объектов, ее библиотека объектов избавляет вас от необходимости беспокоиться о деталях сложной структуры базы данных. Кроме того, вы можете использовать тот же набор объектов для манипулирования базами данных других типов.

Вы можете выбирать среди нескольких библиотек баз данных; чаще других пользуются предложения компании *Microsoft*. Сюда относятся ADO (ActiveX Data Objects — Объекты данных ActiveX), текущий стандарт, и DAO (Data Access Objects — Объекты доступа к данным), старая библиотека объектов, но все еще широко используемая. Другие поставщики предлагают свои собственные библиотеки объектов баз данных. Например, если вы работаете исключительно с файлами dBase, вам следует подумать об использовании CodeBase (продукт компании *Sequiter Software)*.

В данный момент компания *Microsofi* настоятельно рекомендует использовать библиотеку объектов ADO. Эта библиотека предлагает простую в использовании объектную модель, которая одинаково хорошо подходит для работы как с локальными, так и удаленными данными. Она содержит все инструменты, необходимые для одновременной работы с масштабируемыми сетевыми приложениями большого количества пользователей, но также замечательно подходит и для отдельных проектов, используемых на настольных компьютерах. ADO позволяет получать доступ не только к SQL-данным, но и данным других типов, таким как сведения Outlook, которая инструкции SQL не поддерживает.

Установкабиблиотекиобъектовбазыданных

Библиотека ADO поставляется в составе Office XP и Office 2000, в то время как DAO поставлялась с ранними версиями Office, включая Office 97, а также другими VBA-приложениями. Если на вашем компьютере не установлена ни одна из этих библиотек объектов, вы можете загрузить их с Web-узла компании *Microsoft,* что, помимо всего прочего, гарантирует наличие на вашем компьютере самых последних версий этих библиотек.

После того как вы установили необходимую библиотеку объектов баз данных, вы должны указать ее наличие в VBA-проекте, прежде чем сможете использовать ее объекты. Для того чтобы это сделать, воспользуйтесь командой Tools⇨References, чтобы открыть диалоговое окно References (см. главу 12), в котором вам следует добавить необходимые сведения.

Использованиесвязанныхэлементовуправления

Какую бы библиотеку объектов баз данных вы ни выбрали, вы можете значительно расширить свои возможности программиста, используя *связанные* элементы управления AvtiveX, что означает, что они автоматически отображают данные из таблицы базы данных, с которой вы хотите работать. Когда пользователь изменяет данные в одном из подобных элементов управления, изменения автоматически будут отражены и в базе данных без каких-либо действий с вашей стороны.

Возможность связывания элементов управления с данными представляет собой ключевое отличие Access от большинства других VBA-приложений. Однако, если вы используете приложение, например Word, которое не содержит элементы управления соответствующих типов, вы можете добавить их, отдельно приобретя специальные наборы инструментальных средств ActiveX. В состав пакета Microsoft Office Developer входит большое количество элементов управления для связывания данных, которые можно использовать в любом приложении Office, а также целый ряд других инструментов и утилит, значительно упрощающих создание программ управления базами данных на VBA. Кроме того, свои разработки предлагают многие сторонние поставщики.

Несколько связанных технологий баз данных

При программировании баз данных в Office вы столкнетесь еще с целым рядом терминов, поэтому вам стоит ознакомиться с ними заранее. *ODBC* (Open DataBase Connectivity— Открытый интерфейс доступа к базам данных) — это старый стандарт программирования от компании *Microsoft,* предназначенный для организации доступа к структурированным данным любого типа с помощью SQL. ODBC проектировался в те времена, когда в мире программирования царили функции C/C++, что вам совершенно не нужно при работе с VBA. Однако, поскольку стандарт ODBC существовал не один год, ODBC-драйверы доступны для баз данных всех мыслимых типов.

OLE DB (OLE для баз данных) — это новая спецификация *Microsoft* для доступа к данным, которая постепенно должна вытеснить стандарт ODBC. Чисто теоретически, OLE DB

обеспечивает более простой и гибкий доступ к информации любого типа, которая может представляться в табличной форме. Эта информация сохраняется в обычной настольной базе данных, в базе данных на основе SQL на сервере, в виде набора сообщений электронной почты и т.д. Для представления объектов базы данных программисту используется OLE DB, а не ODBC. Однако еще не каждый поставщик баз данных написал соответствующее средство доступа OLE DB Provider, позволяющее "упаковать" информацию базы данных в форму объектов для дальнейшего их использования с помощью ADO. По этой причине ADO поставляется с дополнительным средством доступа OLE DB Provider, способным подключить любую ODBC-базу данных к ADO.

Программирование баз данных: доступные варианты выбора

Как известно, Access представляет собой приложение для работы с базами данных, входящее в состав Microsoft Office, поэтому использование Access для построения пользовательских программ для работы с базами данных кажется вполне логичным. Однако приведенные ниже причины могут подтолкнуть вас к рассмотрению других возможных вариантов.

Основное назначение Access — упрощение взаимодействия рядовых пользователей с ядром базы данных. Пользовательский интерфейс Access содержит массу полезных средств, таких как проектирование запросов в графическом виде, а также настраиваемые формы и отчеты. Независимо от того, создаете вы или используете программу для работы с базами данных, все эти средства значительно упрощают получение сведений из базы данных и их представление на экране. Но, как бы красиво ни выглядели все эти средства, они достаточно далеки от ядра базы данных, которое и выполняет всю основную работу по обнаружению и извлечению необходимых вами данных.

Мое личное мнение таково: поскольку пользовательский интерфейс и ядро базы данных • — это разные части программного обеспечения, вы можете использовать инструменты, отличные от Access, для того чтобы сообщить ядру базы данных, что же именно вы от него хотите, Именно это вы и делаете, когда создаете программу для работы с базами данных, используя VBA. Даже если вы создаете код в рамках Access, полученный VBA-код взаимодействует с ядром базы данных как с независимым программным компонентом, таким как ADO, а не посредством Access.

Возможные варианты разработки баз данных

Если вам *не нужна* Access для управления ядром базы данных, какой же инструмент использовать для создания программы для работы с базами данных? Возможные варианты, а также их преимущества, перечислены ниже.

- ✓ Access. VBA-программы, созданные в среде Access, могут напрямую обращаться к объектам этого приложения, в частности к формам и отчетам. Поскольку формы и отчеты Access обладают встроенными функциональными возможностями баз данных, они позволяют сократить цикл разработки программ значительнее, чем VBA общего назначения. А так как Access предоставляет быстрый доступ к данным, вы всегда можете работать со своими данными, даже если работа над программой еще не закончена. Это замечательная среда для разработки несрочных проектов только в удобное время для вас время.
- ✓ Другое **приложение** Office, такое как **Word** или Excel. В многих пользовательских программах доступ к базам данных представляет собой лишь часть функ-

циональных возможностей. Если программа, которую вам необходимо написать, будет выполнять такие функции, как обработка текста или математические и статические расчеты, создание ее в среде Word или Excel позволит вам напрямую воспользоваться преимуществами соответствующего приложения. С помощью ADO ваша VBA-программа все равно будет обладать возможностью получать, отображать данные и манипулировать ими из баз данных. А программа окажется еще и небольшая, быстрая и менее сложная, чем если вы использовали бы модель COM для автоматизации доступа к данным Access из, скажем, Word (или автоматизации доступа к данным Word из Access).

Сами по себе формы и отчеты VBA не способны взаимодействовать с базами данных, но написать код для связывания формы данных вручную не так уже сложно. Кроме того, вы можете приобрести Office Developer или другие инструментальные средства от сторонних производителей, которые позволят связать формы и элементы управления с данными. Вы можете также переслать результаты своей работы непосредственно в другие VBA-приложения, так как они не зависят от форм Access.

- ✓ **Visual Basic (не VBA).** Visual Basic — это восхитительный инструмент для разработки баз данных. Написанный вами код на Visual Basic может через ADO подключаться к базам данных, как и при использовании VBA, однако формы Visual Basic напрямую поддерживают работу с базами данных, точно так же, как и формы Access. Программа, созданная с помощью Visual Basic, обладает двумя значительными преимуществами перед программой, созданной с помощью VBA-приложения, такого как Word или Excel. Во-первых, она быстрее работает, так как программы, написанные на Visual Basic, компилируются, а не интерпретируются при каждом выполнении, как программы, созданные с помощью VBA. Во-вторых, вы можете свободно распространять программы: конечным пользователям не понадобится базовое VBA-приложение, с помощью которого оно было создано.

Программирование баз данных с помощью Access

Прежде чем вы приступите к программированию базы данных в Access, вам следует ознакомиться со всеми отличиями Access от остальных VBA-приложений. Эти различия осложняют перевод программы, созданной с помощью Access, в другое VBA-приложение.

- ✓ **Формы Access несовместимы со стандартными пользовательскими формами VBA и формами Visual Basic.** Если вы решите, что ваша программа будет лучше работать в другом VBA-приложении, вам придется создавать все формы заново.
- ✓ Access **включает полностью независимую систему программирования баз данных с помощью VBA, которая базируется на использовании объекта DoCmd.** Объект DoCmd содержит все команды, доступные в меню Access. Используя этот объект, вы сможете открывать таблицы, выполнять запросы и отчеты, отображать формы, забавляться с пользовательским интерфейсом... другими словами, выполнять все действия, доступные пользователю, сидящему перед компьютером.

Если вы опытный пользователь Access, но не имеете ни малейшего опыта программирования, объект DoCmd позволит вам постепенно перейти к использованию VBA. Однако этим объясняется и определенная проблема: объект DoCmd жестко привязывает вас к Access. Если же вы изучите стандартный VBA, вы сможете очень легко перейти к другим средам разработки Visual Basic.

Даже если вы выберете "чистый" VBA, вы все равно не обойдетесь без объекта DoCmd при создании программ в рамках Access. Возможно, в связи с тем, что формы Access не являются стандартными формами VBA, для отображения формы в VBA-программе Access вам придется использовать не стандартный метод Show, а метод OpenForm объекта DoCmd.

Написание кода базы данных с помощью ADO

Хотя проектирование правильных SQL-инструкций может оказаться сложным, написание кода базы данных с помощью ADO оказывается совсем несложной задачей. Вам необходимо освоить работу всего с тремя объектами: Connection, Recordset и Command; их методы и свойства реализованы достаточно логично.

Обработка ошибок

Из-за ограниченного объема главы я не могу подробно остановиться на рассмотрении такого чрезвычайно важного вопроса, как обработка ошибок. Однако очень важно включить код обработки ошибок в каждую процедуру базы данных. Подробные сведения о написании кода обработки ошибок на VBA изложены в главе 9.

Добавление ссылки на ADO

Прежде чем вы сможете использовать ADO и ее объекты в VBA-программе, вы должны сначала добавить в своем проекте ссылку на библиотеку объектов ADO. В окне редактора Visual Basic выберите команду Tools⇨References, после чего установите флажок напротив Microsoft ActiveX Data Objects 2.x Library (на момент написания книги последней версией была 2.5).

Установка соединения

Вашей первоочередной задачей при необходимости доступа к *источнику данных* (базе данных или другому репозитарию данных) станет установка соединения с ним. Для организации подключения между вашей программой и данными предназначен объект Connect; ion.

Подключение к открытой базе данных Jet в Access

Это очень важно! Если вы используете Access для написания ADO-кода для ядра базы данных Jet, вам не нужно создавать объект Connection для работы с базой данных Jet, уже открытой в Access, так как Access автоматически выполняет подобные действия за вас. Для обращения к базе данных используйте свойство Connection объекта CurrentProject программы Access. Это позволяет сделать, например, приведенный ниже фрагмент кода:

```
Dim conADOConnection As Connection
Set conADOConnection = CurrentProject.Connection
```

Кроме того, вы можете легко установить подключение с базой данных SQL Server в проекте Access, если вы пишете код VBA в этом же проекте. В этом случае вам необходимо использовать свойство BaseConnectionString объекта CurrentProject, как показано ниже:

```
Dim conADO As New Connection
conADO.ConnectionString = __
     CurrentProject.BaseConnectionString
```

Создание объектов Connection для других баз данных

В других ситуациях вам придется создавать объекты Connection самостоятельно. Для создания объекта Connection просто объявите имя переменной для объекта, после чего *откройте* подключение. Метод Open получает в качестве аргумента *строку подключения,* содержащие различные параметры, которые определяют используемое средство доступа OLE DB Provider и источник данных, с которым вы работаете. Либо же, вы можете сначала задать свойства объекта Connection, соответствующего элементам строки подключения, после чего уже использовать метол Open. Изучите приведенные ниже примеры эквивалентных инструкций, которые создают объект Connection для одной базы данных Jet:

```
' Пример 1
Dim conADOConnection As New Connection, strConnect As String
strConnect = "Provider=Microsoft.Jet.OLEDB.4.0;" _
     & "Data Source=C:\Data\Toys"
conADOConnection.Open strConnect

' Пример 2
Dim conADOConnection As New Connection
With conADOConnection
     .Provider= "Microsoft.Jet.OLEDB.4.0;"
     .Properties("Data Source") = "=C:\Data\Toys"
     .Open
End With
```

Параметры, необходимые методу Open, зависят от используемого вами средства доступа OLE DB Provider; обратитесь к справочной системе, Web-узлу компании *Microsoft* или документации по Office Developer для получения подробных сведений. Ниже приведен соответствующий пример для SQL Server:

```
Dim conADOConnection As New Connection
Dim strConnect As String
strConnect = "Provider=SQLOLEDB; Data Source= Hecate;" _
     & "Initial Catalog = toys; User ID = sa; Password =;"
conADOConnection.Open strConnect
```

Если вы работаете в Access, ADO нельзя назвать универсальным решением по управлению данными. Проекты Access могут взаимодействовать только с базами данных SQL Server, но ни с какими другими средствами доступа OLE DB Provider. Кроме того, использование Access VBA для создания подключений к базе данных SQL Server требует различных параметров, используемых в других средах разработки. В Access свойству Provider должно быть присвоено значение MSDataShape, а свойству DataProvider следует присвоить значение SQLOLEDB.

Подключение налету

ADO позволяет вам работать с объектами баз данных, обходясь без предварительного создания объекта Connection: вы можете связать эти объекты с подключением только тогда, когда придет время заполнить их реальными данными. Либо же вы можете создать подключение неявно в процессе определения объекта Recordset или Command. Однако создание объекта Connection явным образом упрощает ваш код и позволяет связывать одно подключение с несколькими другими объектами.

Работа с объектами Recordset

Давайте поближе познакомимся с объектами Recordset: вы используете при выполнении фундаментальных операций с данными. Объект Recordset — это контейнер, содержащий данные, полученные из источника данных. Как и положено контейнеру, один объект Recordset может содержать различные записи в разное время.

Создание объекта Recordset

После того как вы объявили переменную для объекта Recordset, вы можете немедленно приступить к работе с его свойствами. Однако, в этот момент он существует только "виртуально". Для заполнения пустого контейнера реальными данными прибегните к одному из следующих приемов:

- ✓ собственный метод Open объекта Recordset;
- ✓ метод Execute объекта Command;
- ✓ метод Execute объекта Connection.

Создание объектов Recordset с помощью метода Open

Простейший способ создания объекта Recordset — использование метода Open этого самого объекта. Метод Open хорошо работает в том случае, если вы используете простые инструкции Select для получения необходимых записей. Следующий фрагмент кода демонстрирует настройку объекта Recordset с помощью метода Open:

```
Dim conman As New Connection
Dim rstMan As Recordset
Dim strSQL As String
...(здесь содержится код, используемый для создания объекта
подключения conMan)
strSQL = "SELECT * FROM Toys" 'выбор всей таблицы Toys
Set rstMan.ActiveConnection = conman
rstMan.Open strSQL,, adOpenForwardOnly, adLockReadOnly, _
    adCmdText
```

Обратите внимание на то, что приведенный выше фрагмент кода связывает подключение с объектом Recordset с помощью свойства ActiveConnection последнего. Кроме того, обратите внимание на то, что параметры, управляющие поведением объекта, указаны в качестве аргументов метода Open.

Создание объектов Recordset с помощью объекта Command

Инструкции SELECT языка SQL подходят далеко не всегда. В приложениях клиент/сервер эффективность очень часто диктует необходимость создания объектов Record-

set путем выполнения процедур (запросов). Если подобной процедуре требуются определенные параметры, в этой ситуации оказывается удобным использование объекта Command для создания объекта Recordset.

Прежде всего настройте объект Command, присвоив его свойству ActiveConnection значение, соответствующее необходимому подключению. После этого вы можете уделить внимание и объекту Recordset. В этом случае вы должны сначала определить параметры этого объекта как свойства. После этого вы наполняете объект Recordset записями, пользуясь результатами выполнения метода Execute объекта Command. Соответствующий пример приведен ниже:

```
Dim conTest As New Connection
Dim cmdTest As New Command
Dim rstTest As Recordset
Dim strSQL As String
...(код, создающий объект подключения contest
... и определяющий строку strSQL)
' Создание объекта Command:
    With cmdTest
        Set .ActiveConnection = contest
        .CommandText = strSQL
        .CommandType = adcmdText
    End With
rstTest.CursorType = adOpenForwardOnly
rstTest.lockType = adLockReadOnly
Set rstTest = cmdTest.Execute()
```

Создание объектов Recordset с помощью объекта Connection

Последний прием для создания объектов Recordset заключается в использовании метода Execute объекта Connection. Этот подход намного проще, чем использование объекта Connection, кроме того, он позволяет вам работать с сохраненными процедурами. Однако, если для выполнения процедур необходимы определенные параметры, вам придется включить эти параметры в инструкцию SQL. Все это проиллюстрировано на следующем примере кода:

```
Dim conVert As New Connection
Dim rstVert As Recordset
Dim strSQL As String
...(код, создающий объект подключения conVert
... и определяющий строку strSQL)
rstVert.CursorType = adOpenForwardOnly
rstVert.lockType = adLockReadOnly
Set rstVert = conVert.Execute()
```

Создание подключения на лету

Если вам известно, что подключение необходимо только для одного объекта Recordset, предварительное создание подключения не обеспечит никаких преимуществ. Вместо этого вы можете указать строку подключения в качестве второго аргумента метода Open объекта Recordset, как показано ниже:

```
Dim rstInPeace As New Recordset
Dim strSQL As String, strConnect As String
strSQL = "SELECT * FROM Bicycles" 'Получение всех записей
' Код, назначающий строку подключения переменной strConnect
rstInPeace.Open strSQL, strConnect, adOpenForwardOnly
```

Определение параметров объектов Recordset

Вы получаете контроль над созданным объектом Recordset с помощью различных параметров, определяющих вид указателя, тип блокировки и т.д. Вы можете определить эти параметры одним из двух способов, в зависимости от того, каким образом вы создавали объект Recordset — с помощью аргументов метода Open объекта Recordset, а также свойств этого объекта. Оба подхода были продемонстрированы с помощью фрагментов кода раньше в настоящей главе

Выбор типа курсора

В базах данных *курсор* означает функциональные возможности, необходимые для перемещения между записями. Тип курсора, который вы выбираете для объекта Recordset, определяет, насколько свободно пользователь сможет работать с записями, а также будут ли автоматически отражаться изменения, внесенные другими пользователями. Используйте свойство CursorType или соответствующий аргумент метода Open для указания выбранного вами варианта. Все доступные варианты перечислены в табл. 17.1. По умолчанию тип курсора определен как Forward-only.

Таблица 17.1. Типы курсоров, доступные для объекта Recordset

Тип курсора	Константа-значение свойства CursorType	Определение
Forward-only	adOpenForwardOnly	Разрешает перемещение между элементами объекта только в одном направлении, или на определенное количество записей, или к последней записи. Изменения, внесенные другими пользователями, не появляются до тех пор, пока набор записей не будет закрыт, а затем снова открыт. Этот тип курсора обеспечивает максимальное быстродействие, но только в том случае, если вам необходимо перемещаться по записям всего один раз
Static	adOpenStatic	Фиксированный набор записей, который не может быть обновлен и не отражает изменения, внесенные другими пользователями, до тех пор, пока он не будет закрыт, а затем снова открыт. Этот вариант подходит для поиска данных и создания отчетов, обеспечивая более высокую скорость, чем варианты Keyset и Dynamic
Keyset	adOpenKeyset	Набор, чьи записи их значения могут изменяться в результате внесения соответствующих изменений в базу данных. Однако подобный набор записей не отражает изменения, внесенные другими пользователями
Dynamic	adOpenDynamic	Этот вариант напоминает набор записей Keyset, за исключением того, что он не отражает изменения, внесенные в базу данных

Выборрасположениякурсора

Свойство CursorLocation объекта Recordset позволяет вам определить, будет ли курсор находиться на компьютере пользователя (компьютере клиента) или на сервере. Для этого присвойте ему значение adUseClient или adUseServer соответственно. Вообще, используйте курсоры на стороне клиента при работе с базами данных SQL Server и другими сетевыми базами данных, а курсоры на сервере — при работе с базами данных Jet. Вы можете определить расположение курсора по умолчанию для всех наборов записей подключения с помощью свойства CursorType объекта Connection.

блокирование

Свойство LockType объекта Recordset определяет, как именно должна вести себя программа в том случае, если два или больше пользователей пытаются одновременно изменить одну и ту же запись. По умолчанию значение этого свойства равно adLockPessimistic, что предотвращает внесение изменений. Если вы хотите иметь возможность изменять набор записей, вы должны присвоить свойству LockType одно из значений, приведенных в табл. 17.2.

Таблица 17.2. Параметры блокировки для объектов Recordset

Тип блокировки	Константа-значение свойства LockType	Определение
No lock (Блокировка отсутствует)	adLockOptimistic	Несколько пользователей могут одновременно изменять одну и ту же запись, однако при этом получают уведомление о том, что внесенные ими изменения могут конфликтовать с изменениями, внесенными другими пользователями (записи блокируются только при выполнении метода Update). Этот вариант хорошо подходит для баз данных, с которыми работает только один пользователь, а также многопользовательских баз данных, в которых всегда правильными и полными считаются сведения, добавленные самыми последними
Batch update (Пакетное обновление)	adLockBatchOptimistic	Этот вариант напоминает предыдущий, за тем исключением, что записи блокируются при групповом, а не индивидуальном обновлении
Read only (all records) (Только чтение (все записи))	adLockReadOnly	Все записи блокируются в то время, когда набор записей открыт; изменять их не может ни один пользователь, Однако доступ к записям разрешен
Read only (edited records) (Только чтение (изменяемые записи))	adLockPessimistic	Блокируется только та запись, с которой пользователь работает в данный момент. Блокировка снимается в тот момент, когда пользователь переходит к другой записи

Проверка записей

Если при создании объекта Recordset с помощью VBA не возникло никаких проблем, вашим первым действием должна быть проверка того, что полученный набор вообще содержит какие-то данные. Если это так, сообщите пользователю о том, что для работы нелоступна ни одна запись. Прием заключается в проверке значений свойств BOF (Beginning Of File — Метка начала файла) и EOF (End Of File — конец файла). Если значения обоих свойств равны True, набор записей пуст. Лучший способ организации подобной проверки — использование логического оператора And, который возвращает значение True только в том случае, если оба логических выражения равны True, как показано в приведенном ниже примере:

```
If rstY.30F and rstY.EOF Then
     MsgBox "Записи в этом наборе отсутствуют!"
End If
```

Перемещение по наборам записей и нахождение определенных записей

ADO позволяет вам перемещаться по набору записей достаточно легко. Например, для перемещения к первой записи в наборе используется метод MoveFirst, для перемещения к последней записи в наборе — метод MoveLast, а для перемещения к следующей или предыдущей записи в наборе — метод MoveNext или MovePrevious соответственно. Метод Move позволяет вам перемещаться на определенное количество записей в наборе вперед или назад. Например, инструкция rst Z . Move -3 перемещает на три записи назад.

Если вы точно знаете, что захотите вернуться к определенной записи в дальнейшем, создайте для этой записи закладку. При работе с записью присвойте переменной значение свойства Bookmark объекта Recordset, как показано ниже:

```
varBookmark1 = rstA.Bookmark
```

После этого вы можете вернуться к этой закладке в дальнейшем, просто обратив инструкцию:

```
rstA.Bookmark = varBookmark1
```

Метод Seek, а также четыре метода Find (FindFist, FindLat, FindNext и FindPrevious) позволяют вам отследить определенную запись, базируясь на ее содержимом. Поскольку метод Seek обнаруживает целевую запись, используя индексный номер, он работает намного быстрее, чем методы Find, однако, прежде чем вы сможете его использовать, ссдержимое базы данных должно быть проиндексировано.

Добавление и удаление записей

Используйте метод AddNew для занесения новой записи в набор или перехода к новой записи. Если вы уже создали новый объект Recordset под названием rstIng, все, что вам необходимо для добавления новой записи, — это инструкция rst.Ing. После этого вы можете приступать к заполнению полей записи (подробности в разделе "'Изменение данных поля" дальше в настоящей главе). Однако, если хотите, можете указать значения полей сразу при создании новой записи. Метод AddNew позволяет вам указывать поля и их значения, как показано ниже:

```
With rstlng
     .AddNew Array( "Имя", "Возраст", "Пол"; , _
```

```
        Array("Анна", 42, "Ж")
End With
```

Как видно из этого примера, вы передаете аргументы методу AddNew в виде пары массивов, первый из которых содержит имена полей, а второй — их значения. В этом примере я использовал функцию Array VBA для создания на ходу массива, содержащего строковые значения. Вместо этого вы можете использовать переменные, представляющие массивы, которые вы создали раньше. В любом случае подобный прием для обеспечения значений полей массивов в момент создания новой записи означает, что вам не нужно вводить отдельную строку кода для каждого поля.

Для удаления текущей записи предназначен метод Delete.

Чтение полей данных

Работа с текущим значением определенного поля с помощью программного кода так же проста, как и считывание значения свойства Value этого поля. Укажите поле по имени или по его индексному номеру, как показано в приведенных ниже примерах. Обратите внимание на то, что свойство Value является свойством по умолчанию, поэтому указывать его в коде не обязательно:

```
If rstYGate.Fields("Service visits").Value >10
     MsgBox "This unit needs a major overhaul!"
End If

strCurrentFieldData = rstYGate.Fields(3)
```

Поскольку коллекция Fields является коллекцией по умолчанию объекта Recordset, вам не обязательно указывать ее по имени. Для обращения к полю просто укажите его имя, предварительно поставив восклицательный знак и заключив в квадратные скобки имена, содержащие пробелы:

```
rstYGate!Date = #5/15/2001#

With rstYGate
     intItems =![Oil cans]
End With
```

Изменение данных поля

Изменение (обновление) данных поля определенной записи с помощью ADO совсем несложно. Все, что вам необходимо сделать, — указать новое значение, после чего переместиться к другой записи, как показано ниже (здесь предполагается, что набор записей rstBucket уже открыт и содержит определенные данные):

```
With rstBucket
     .Fields(0).Value = "Love"
     .MoveNext
End With
```

А если вы не хотите перемещаться от текущей записи, можно внести изменения в базу данных, воспользовавшись методом Update. В приведенном ниже примере я воспользовался преимуществами, обеспечиваемыми стандартным состоянием коллекции Fields и свойством Value, чтобы свести к минимуму код, который необходимо написать:

```
With rstBucket
     !Value =8.93
```

```
    .Update
End With
```

Как и метод AddNew, метод Update позволяет вам передать значения новых полей с помощью пары массивов, как показано ниже:

```
With rstBucket
     .Update Array("Имя", "Звание", "Любимый напиток"),_
          Array("Лола", "Младший лейтенант", "Кофе")
End With
```

Повторение операций с несколькими записями

Используйте цикл Do для проверки или операций с несколькими записями в наборе, как показано в следующем примере;

```
'Циклическое перемещение между записями в наборе
     With rstInPeace
          Do Until .EOF
               Debug.Print.Fields(0)
               .MoveNext
          Loop
     End With
     SetrstInPeace = Nothing
End Sub
```

Использование объекта Command

В ADO объект Command представляет команду, такую как инструкция SQL или сохраненная процедура, которая может применяться к источнику данных. Вы можете использовать объекты Commands для получения записей из объектов Recordset, а также выполнения таких операций, как обновление или удаление нескольких записей. (Пример программного кода, приведенный ниже, выполняет только последние операции; примеры, использующие наборы записей, приведены раньше в настоящей главе.)

Не удивляйтесь, если вы не сможете использовать объект Command с определенным источником данных. Средства доступа OLE DB Provider не нужны для реализации объекта Command, а значит, и не требуются для обработки параметров.

О хранимых процедурах

Хранимые процедуры — это запросы и другие операции, которые вы или кто-нибудь еще определили раньше (обычно с помощью SQL) и сохранили в источнике данных. В качестве хранимых процедур можно привести запросы Access. Несмотря на то, что вы можете спроектировать запрос Access визуально с помощью сетки, он представляет собой инструкцию, которую вы увидете, выбрав представление SQL. Запрос сохраняется как часть файла базы данных .mdb, к которой он относится. Сетевые базы данных, такие как SQL Server, также позволяют вам определять подобные хранимые процедуры.

Хранимая процедура уже готова к использованию, поэтому вам необходимо знать только ее имя: вы можете спокойно забыть обо всех тонкостях определения запроса с помощью программного кода. Еще важнее, что хранимые процедуры выполняются намного быстрее и более надежны при использовании в сети, чем эквивалентные инструкции SQL. Однако инструкции SQL для вашей хранимой процедуры не являются частью кода, а значит, вы не сможете при необходимости их изменять.

Настройка объекта Command

Для настройки объекта Command вам следует начать с объявления соответствующей переменной и создания собственно объекта. После этого вы можете использовать свойства объекта для связи его с подключением, для определения команд, которые должны выполняться в форме инструкции SQL или имени хранимой процедуры, а также для определения типа операции. После этого вы можете использовать метод Execute объекта Command для действительного выполнения команды. Ниже приведен пример инструкции обновления SQL. Обратите внимание на свойство CommandType, которому вы должны присвоить значение adCmdText для передачи инструкции SQL источнику данных:

```
Jim consecrate As Connection
Dim cmdVBA As Command
Dim prmDate
(код настройки объекта Connection...)
Set cmdVBA = New Command
With cmdVBA
      .ActiveConnection = consecrate
      .CommandText = _
           "UPDATE Bicycles SET OnSale = True" _
                & "WHERE Category = 4;"
      .CommandType = adCmdText
      .Execute
End With
```

Использование параметров команды

Если для выполнения команды требуются *входные параметры* (значения, передаваемые команде при выполнении, например диапазон данных или критерий поиска), вы должны определить индивидуальные объекты Parameter, добавляя их к коллекции Parameters объекта Command. Пример, приведенный ниже, иллюстрирует, как это все работает, и как можно использовать объект Command для выполнения хранимой процедуры, а не инструкции SQL:

```
Dim conSecrate As Connection
Dim cmdVBA As Command
Dim prmDate
(код для настройки объекта Connection...)
Set cmdVBA = New Command
With cmdV3A
      .ActiveConnection = consecrate
      .CommandText = "qryDeleteOldRecords"
      .CommandType = adCmdStoredProc ' in Jet, _
           adCmdTable
End With

Set prmDate = New Parameter
With prmDate
      .Name = "Date"
      .Value = InputBox "Enter the cut-off date."
      .Type = adDate
      .Direction = adParamInput
End With
With cmdVBA
```

```
    .Parameters.Append prrnDate ' Добавление параметра
    .Execute ' Выполнение команды
End With
```

Если вы выполняете запрос, сохраненный в базе данных Jet/Access, используйте значение `adCmdTable` свойства `CommandType` объекта Command, а не `adCmdStoredProc`, относящееся к SQL Server и многим другим серверам баз данных.

Работа с SQL

Достаточно просто открыть набор записей с помощью ADO, но заполнение его данными — задача гораздо сложнее. Если вы намерены создавать серьезные приложения баз данных, вам следует изучить SQL, а также познакомиться с инструментами, автоматизирующими написание инструкций SQL.

Как избежать SQL

Добавление инструкций SQL в код VBA чем-то сродни изучению письма, содержащего отрывки на греческом языке, сложные математические вычисления и музыкальное представление. Несмотря на то, что SQL более компактный и узконаправленный язык программирования, чем Visual Basic, многие VBA-программисты находят его сложным в использовании по той причине, что он просто другой. Осознавая подобное положение, вы можете захотеть использовать инструменты разработки, которые будут составлять инструкции SQL, базирующиеся на вашем выборе из списка полей, критериях поиска определенных полей, а также действиях, которые должен совершать запрос.

Встроенный конструктор запросов Access — один из подобных инструментов. После того как вы создали и протестировали запрос Access, у вас есть два варианта выбора для добавления объекта `Command` в ваш код VBA для выполнения запросов. Объект Command может выполнять запрос как хранимую процедуру или же вы можете копировать SQL-код, полученный с помощью конструктора запросов, а затем вставить его в свой код VBA. Для того чтобы воспользоваться последним приемом, сконструируйте запрос в Access и тщательно его проверьте; затем выберите команду **View⇨SQL View (Вид⇨Режим SQL),** чтобы отобразить соответствующую инструкцию. Скопируйте ее в буфер обмена, переключитесь к редактору VBA, после чего вставьте, заключив в кавычки, в инструкцию VBA, которая определяет значение свойства `CommandText` объекта `Command`. Подробные сведения о выполнении инструкций SQL и хранимых процедур изложены в разделе "Использование объекта Command" раньше в настоящей главе.

Конструктор запросов Access замечательно справляется с поставленными задачами, но он не может обработать все встречающиеся ситуации. Он не позволяет создавать запросы сложных типов, т.е. подзапросы. Конечно же, его хранимые процедуры работают только с базами данных Jet (однако другие базы данных содержат подобные средства визуального конструирования запросов). Вы можете попробовать конструктор запросов Access для получения SQL-кода для баз данных другого типа, однако не забывайте о том, что разные базы данных используют разные версии SQL, поэтому вам придется вносить коррективы в полученный код.

Знакомство с диалектами SQL

Хотя SQL и считается практически универсальным языком создания запросов для систем управления базами данных, многие базы данных понимают только определенные диалекты SQL. Jet, например, предлагает несколько нестандартных усовершенствований SQL, но не реализует некоторые функции, присутствующие в стандартной версии SQL. В настоящей главе я ограничусь рассмотрением Jet-версии SQL, поскольку именно с ней сможет работать каждый пользователь Office. Если вы используете другую систему управления базами данных, нам придется подгонять изложенные мною сведения под доступный вам диалект SQL.

Вставка инструкций SQL в VBA-код

Фрагменты кода, приведенные раньше в настоящей главе, должны были проиллюстрировать, как добавлять инструкции SQL в код, написанный на VBA. Здесь очень важно не забывать о том, что VBA обрабатывает инструкции SQL как текстовые строки, а не как часть собственно программного кода. Эти строки претерпевают специальную обработку, когда интерпретируются как аргументы методов Open или Execute объектов ADO, но до тех пор они остаются обычными строками VBA, содержащими текст.

По этой причине вам следует заключать каждую инструкцию SQL в двойные кавычки. Независимо от того, настраиваете вы свойство CommandText объекта Command или указываете аргумент метода Open объекта Recordset, использование двойных кавычек обязательно. И опять, вы найдете много примеров в предыдущих разделах настоящей главы. И не пропустите материал раздела "Настройка набора записей: задаем критерии" дальше в настоящей главе.

Написание инструкций SELECT

Процесс создания объектов Recordset модели ADO я подробно рассмотрел раньше в настоящей главе. В этом разделе основной акцент будет сделан именно на коде SQL, необходимом для определения того, какие записи относятся к набору. Вам необходимо выполнить подобную задачу как можно точнее, поэтому инструкция SELECT — это именно то, что вам нужно.

Простейшая форма инструкции SELECT получает все записи из одной таблицы. Приведенный ниже пример возвращает все поля и все записи из таблицы Toys:

```
SELECT * FROM Toys
```

Поскольку эта инструкция не содержит никаких дополнительных критериев, в возвращаемый в результате выполнения инструкции набор записей включаются все записи из таблицы Toys. Звездочка означает, что набор записей, помимо всего прочего, содержит еще и все поля из таблицы.

Инструкции SELECT извлекают записи из базы данных, но не изменяют при этом хранящиеся там данные. Для внесения подобных изменений вы должны изменить значения полей в наборе записей, после чего использовать такие инструкции SQL, как UPDATE или DELETE.

Связывание нескольких таблиц в инструкциях SELECT

Одна инструкция SELECT может работать с более чем одной таблицей. Просто перечислите все необходимые таблицы, с которыми должна работать инструкция, как показано ниже:

```
SELECT * FROM Toys, Clerks
```

Однако набор записей, полученный в результате подобного перечисления таблицы, оказывается не таким уж полезным. Ничто не связывает эти таблицы, поэтому база данных не знает, какая запись из первой таблицы соответствует какой записи из второй таблицы. Получается, что каждой записи из первой таблицы соответствуют все записи из второй.

Для того чтобы правильно связать две таблицы, выполните в инструкции SELECT *соединение. Внутреннее соединение,* наиболее распространенное, создает запись в наборе, базируясь на соответствии записей в исходных таблицах, а соответствие базируется на одинаковых значениях определенных полей таблиц. Например, приведенная ниже инструкция создает набор записей, в котором перечисляются запасные компоненты для каждой игрушки в списке:

```
SELECT Toy, Rep From Toys INNER JOIN Reps On Toys.ID =
    Reps.ToyID
```

Для создания внутреннего соединения разместите конструкцию INNER JOIN между именами таблиц в инструкции FROM. После этого укажите ключевое слово ON, которое определяет поля, содержащие сравниваемые значения. Обычно имена полей после ключевого слова ON разделяются знаком равенства, что свидетельствует о том, что для выбранных записей значения, содержащиеся в указанных полях, должны совпадать (допускается использование и других операторов сравнения).

Выбор полей

Для указания набора полей укажите их имена явным образом, как показано ниже:

```
SELECT Toy, InStock, OnOrder FROM ToyInventory
```

Если имя поля содержит пробелы или знаки пунктуации, заключите его в квадратные скобки, как показано ниже:

```
SELECT Toy, [List Price], [Sale Price] FROM ToyInventory
```

По умолчанию значение свойства Name каждого объекта Field в наборе записей, полученном с помощью инструкции SELECT, равно имени соответствующего поля в исходной таблице. Вы можете назначить другие имена полей (псевдонимы) в наборе записей, используя ключевое слово AS для каждого поля, которое вы решили переименовать:

```
SELECT Toy AS ToyName, InStock AS OnHand, OnOrder FROM ToyInventory
```

Если вы извлекаете записи из нескольких таблиц и хотите выбрать поля с одинаковыми именами, укажите перед именем поля имя соответствующей таблицы. Вот соответствующий пример:

```
SELECT ToyInventory.Name, Clerks.Name FROM ToyInventory, Clerks
```

Создание вычисляемых полей и значений

Вы можете построить набор записей, который будет содержать новые поля, а их значения вычисляются на базе значений, содержащихся в базе данных. В инструкции SELECT подобные поля определяются с помощью выражений, базирующихся на операторах и функциях VBA. Например, предположим, что вам необходимо перечислить цены на товары, которые получатся после снижения на 10%:

```
SELECT Toy, (Price * .9) AS SalePrice FROM ToyInventory
```

Обратите внимание на то, что при определении вычисляемого поля вы должны включить ключевое слово для создания псевдонима (имени) поля в наборе записей. Скобки необязательны, однако они помогают подчеркнуть выражение, которое необходимо вычислять. В выражении, позволяющем получить значение нового поля, допускается использование нескольких полей, например, так: (Price * InStock) AS InventoryValue.

В качестве еще одного примера предположим, что по какой-то причине вам необходимо получить набор записей, в котором будут перечислены имена всех клерков прописными буквами, но при этом способ представления имен в самой базе данных не должен изменяться. Это позволяет сделать приведенная ниже инструкция:

```
SELECT UCase(Name) AS [Clerk's name] FROM Clerks
```

При использовании *обобщенных функций* SQL инструкция SELECT позволяет получить набор записей, который будет содержать только одно результирующее значение, например, количество записей, содержащих заданное значение в определенном поле:

```
SELECT Count(Recyclable) AS [Can Recycle] From Toys
```

или среднее значение всех полей:

```
SELECT Avg(Price) AS [Average Price] FROM Toys
```

После этого вы можете передать значение поля этой записи переменной в вашем коде VBA для использования в вычислениях или отображения в форме:

```
intRecyclableCount = rstRecyclableToys![Can recycle]
```

К обобщенным функциям относятся следующие: Count, Avg, Sum, Min, Max, а также несколько статистических функций.

Выбор записей с помощью предикатов DISTINCT, DISTINCTROW и TOP

Используйте *предикаты* DISTINCT, DISTINCTROW и TOP в инструкции SELECT в качестве простых инструментов получения определенных наборов записей из баз данных. Эти специальные слова необходимо указывать сразу после инструкции SELECT, как показано в табл. 17.3.

Таблица 17.3. Предикаты SQL для выбора записей

Предикат	Использование	Пример
DISTINCT	Выбирает только одну запись, если база данных содержит несколько записей с идентичными данными в указанных полях	`SELECT DISTINCT Address FROM Members` Возвращает набор записей, содержащий только одну запись для каждого адреса, даже если в таблице `Members` содержится по несколько записей для каждого адреса
DISTINCTROW	Выбирает все уникальные записи, базируясь на значениях всех полей. Если две записи отличаются всего одним символом, они все равно будут включены в набор; если они полностью идентичны, в набор будет включена только одна из них	`SELECT DISTINCTROW Name, Address FROM Members` Возвращает набор записей, содержащий записи с полями `Name` и `Address`. Набор может содержать дублирующиеся записи, но только в том случае, если отличаются значения других полей

Окончание табл. 17.3

Предикат	Использование	Пример
TOP л	Выбирает указанное число записей в верхней или нижней части диапазона, определяемого ключевым словом ORDER BY	SELECT TOP 10 ToyName FROM Toys ORDER BY UnitsSold Возвращает набор записей, содержащий сведения о 10 лучше всего продаваемых игрушках. Для определений 10 хуже всего продаваемых игрушек вам следует добавить ключевое слово ASC (ascending - по убыванию) после слова UnitsSold

Настройка набора записей: задаем критерии

Для ограничения набора записей только теми записями, которые удовлетворяют определенным критериям, добавьте к инструкции SELECT ключевое слово WHERE, как показано в приведенных ниже примерах:

```
SELECT * FROM Toys WHERE Price <= 20
SELECT Customer, Date FROM Sales WHERE Date = #10/24/2000#
SELECT Name, Rank, CerealNumber FROM Kids WHERE Rank = 'Queen'
SELECT Name, Age, [Shoe Size] FROM Kids WHERE Age Between 3 And 6
```

Как легко видеть, ключевое слово WHERE **указывается после ключевого слова** FROM **и содержит выражение, определяющее критерий, которому должны соответствовать записи, чтобы попасть в набор. Кроме того, эти выражения не похожи на обычные выражения VBA. Во-первых, строковые значения заключаются в одинарные, а не двойные кавычки. Во-вторых, вы можете определять диапазоны с помощью конструкции** Between ... And, **которая в VBA отсутствует. И в SQL оператор** Likeфункционирует совсем не так, как в VBA.

Вы можете объединить несколько выражений, используя логические операторы (And, Or **и т.д.), как показано ниже;**

```
SELECT * FROM Toys WHERE Price > 20 And Category = 'Action Figures'
```

В коде VBA принято использовать одинарные кавычки для определения строки в инструкции SQL, которая целиком является строкой с точки зрения VBA, а значит, заключается в двойные кавычки. Например, вы можете настроить объект Command **следующим образом:**

```
strSQL = "SELECT Name FROM Kids WHERE Hates = _
    'Brocolli'"
cmdEr.CommandText = strSQL
```

Очень часто, особенно при использовании ключевого слова WHERE, **вам необходимо, чтобы часть инструкции SQL основывалась на переменной; например, если вы выполняете запрос, базирующийся на данных, введенных пользователем в текстовом поле формы. Добавьте значение переменной к остальной части строки. Если переменная представляет строковое значение, не забудьте заключить ее в одинарные кавычки, как показано в следующем примере:**

```
strSQL = "SELECT Name FROM Kids WHERE Hates = ' " _
& frmInputForm.TextBox1 & "'"
```

Если переменная представляет данные, а не строку, заключите ее между символами #, а не в одинарные кавычки. Переменные, представляющие числовые значения, не требуют использования каких-либо открывающих и закрывающих символов.

Группирование записей

Ключевое слов GROUP BY позволяет вам объединять записи, содержащие одинаковые значения в указанных полях, преобразуя их в одну запись в полученном наборе записей. Обычно это ключевое слово используется в том случае, если вам необходим набор записей, содержащий общие сведения о данных. Например, вам понадобилось узнать, сколько записей содержится в базе данных для каждого значения указанного поля. Соответствующий пример приведен ниже:

```
SELECT Category, Count([Category]) AS [Number of Items] FROM Toys
GROUP BY Category;
```

В результате выполнения этой инструкции можно получить набор записей примерно такого вида.

Категория	Количество элементов
Солдатики	42
Куклы	37
Игры	29
Мягкие игрушки	23
Головоломки	17
Спортивные товары	31

Вы можете использовать другие обобщенные функции SQL, такие как Max или Avg, для получения других сведений.

Дополнительный отбор с помощью ключевого слова HAVING

Ключевое слово HAVING следует после ключевого слова GROUP BY и позволяет определить критерии для сгруппированных записей. Оно работает практически так же, как и ключевое слово WHERE; вы можете использовать его отдельно или в комбинации со словом WHERE для того, чтобы наложить на полученные записи дополнительные ограничения. В этом примере ключевое слово HAVING включает только те категории, которые содержат как минимум пять записей, удовлетворяющих критериям, определенным с помощью ключевого слова WHERE:

```
SELECT Category, Count(Category) As [Number cf Items] FROM Toys
   WHERE Price > 100 GROUP BY Category
   HAVING Count(Category) > 4
```

Сортировка с помощью ключевого слова ORDER BY

Используйте оператор ORDER BY для сортировки записей, полученных с помощью инструкции SELECT, в соответствии со значениями одного или нескольких полей. Оператор ORDER BY указывается в конце инструкции, как показано ниже:

```
SELECT Toy, Price, InStock FROM ToyInventory ORDER BY Toy
```

В полученном наборе записей список игрушек будет упорядочен по именам.

Для того чтобы упорядочить этот список по иене, вам следует использовать инструкцию, приведенную ниже. Как видите, можно проводить сортировку по значениям нескольких полей, указав эти поля в необходимом порядке сортировки:

```
SELECT Toy, Price FROM Toylnventory ORDER BY Price DESC, Toy
```

По умолчанию сортировка всегда проводится по возрастанию. Для явного указания порядка сортировки используйте ключевое слово DESC (descending — убывание) или ASC (ascending — возрастание), после которого необходимо указать имя соответствующего поля. В приведенном выше примере сортировка проводилась по убыванию, поэтому товары с максимальной ценой указаны в списке первыми.

Выполнение групповых обновлений и удалений в SQL

Инструкции UPDATE и DELETE позволяют вам изменять или удалять группу записей в источнике данных с помощью всего одной команды. Эти инструкции работают непосредственно с исходной базой данных; вам не нужно предварительно получать набор записей, изменять записи, а затем переносить в базу данных внесенные изменения. Изучите приведенный ниже пример, иллюстрирующий повышение цены на 10% для товаров определенной категории:

```
UPDATE Toys SET Price = Price * 1.1 WHERE Category = 'Trains'
```

Имя таблицы, с которой вы работаете, указывается сразу после слова UPDATE. После этого указывается оператор SET, с помощью которого вы определяете значение одного или нескольких полей в таблице. И наконец, необязательный оператор WHERE позволяет вам задать критерии, ограничивающие записи, к которым будут применяться изменения. Оператор WHERE работает точно так же, как и в инструкциях SELECT.

Инструкция DELETE еще проще в использовании, чем инструкция UPDATE: ее действия сводятся к простому удалению записей. Приведенный ниже пример удаляет записи для всех игрушек, которые отсутствуют на складе и не были заказаны:

```
DELETE FROM Toys WHERE InStock = 0 And OnOrder = 0
```

Для удаления значений отдельных полей, а не целых записей, используйте инструкцию UPDATE вместе с оператором SET, определяющим значение поля равным Null.

Инструкции UPDATE и DELETE приводят к необратимым изменениям в базе данных; вы не сможете отменить действие этих инструкций. Поэтому, прежде чем выполнять любую из этих инструкций, обязательно создайте резервную копию базы данных.

Глава 18

Работа с файлами на диске

В этой главе...

- Получение доступа к файлу по номеру
- Выбор режима доступа к файлу
- Чтение и запись в файлы с использованием не объектно-ориентированных методов

В дополнение к объектно-ориентированным методам работы с файлами данных, описанным в главе 12, VBA предлагает альтернативный метод для чтения и записи данных с дисковых файлов. Эта старая система по-прежнему обладает определенными достоинствами. Хоть вы и не можете работать со свойствами файла и методами, вы получаете более полный контроль над тем, как данные организуются в файле, и над тем, какие данные читаются и записываются. В отличие от объектно-ориентированной файловой системы операторы и функции, описанные здесь, встроены в VBA и не требуют внешних библиотек.

Номер — это ключ,

После того как файл открыт для доступа VBA, обращение к нему осуществляется по номеру, а не по имени. Если приходится работать с несколькими файлами одновременно, отслеживать то, какой файл вам необходим в данный момент, совсем не просто. В принципе, никто не запрещает работать таким образом. Но дальше в данной главе, в разделе "Не идите на поводу у номеров", показан простой способ решения данного вопроса. Для работы с содержимым файла необходимо его открыть. Для этого используется оператор Open (Открыть). Вот его синтаксис в простейшей форме:

```
Open "pathname" For mode As filenumber
```

А вот пример типичного оператора Open:

```
Open "C:\Мои документы\Мои данные.dat" For Binary As #1
```

Оператор использует три аргумента.

- ✓ pathname (путь). Полный путь (с указанием диска и каталога), определяющий файл, который должен быть открыт как строчное выражение. При вводе строки текста необходимо заключать ее в кавычки, как это сделано в приведенном выше примере. Вместо этого можно использовать строку переменных, а также любое выражение, значение которого будет правильной строкой VBA.
- ✓ mode (режим). Ключ VBA, определяющий то, каким образом вы планируете работать с файлом; подробности — в разделе "Выбор режима доступа к файлу".

- ✓ `filenumber` (номер файла). Аргумент `filenumber` принимает целочисленное значение от 1 до 511, если оно не было присвоено другому файлу. Традиционно, но не обязательно, в номере перед числом помешают знак #. В дальнейшем присвоение номеров не обязательно, однако при желании это можно делать (можно воспользоваться функцией `FreeFile`, которая возвращает следующее доступное значение `filenumber`).

Оператор Open может принимать еще некоторые опциональные аргументы. Все они здесь не рассматриваются, но в следующем разделе рассмотрен один из них — `Len=reclength`, необходимый при работе с файлами произвольного доступа.

Выбора режима доступа к файлу

При открытии дискового файла с помощью оператора Open необходимо включить в него аргумент mode. Данный аргумент говорит VBA об организации файла и о том, каким образом планируется осуществлять доступ к информации. Хотя в качестве аргумента mode может быть любое из пяти возможных ключевых слов, в действительности выбор проводится из трех основных режимов. Каждый из режимов имеет свои достоинства и недостатки, описанные ниже.

Режим доступа к файлу в операторе Open	Описание работы	Ключевые слова
Произвольный доступ	Сохраняет данные в записях идентичной длины, так что чтение или запись могут вестись до тех пор, пока известно положение в файле, Запись в режиме произвольного доступа соответствует любому типу данных VBA фиксированной длины или определенному пользователем типу, содержащему исключительно типы данных фиксированной длины	Random
Последовательный	Данные сохраняются в виде серии символов. Подходит для текстовых файлов, а также для файлов баз данных в кодировке ASCII, использующих разделители-запятые	`Input`(для чтения данных из файла); `Output` (для записи данных в файл); `Append` (для записи данных в конец файла)
Двоичный	Данные сохраняются в строчной форме, в любом заданном порядке	Binary

Не идите на поводу у номеров

Хотя в VBA рекомендуется обращаться к открытым файлам по номеру, запоминать то, с чем связаны какие-то числа, достаточно трудно и неудобно. Вместо номеров файлов можно использовать константы или переменные с понятным названием. Если наперед известен номер, который будет использоваться для отдельного файла, объявите для него константу. Данную константу затем можно использовать в операторе, воздействующем на файл. Вот пример:

```
Const PetsFile = 1, BdayFile = 2
Open "C:\MiscData\PetsLog.txt" For Input As PetsFile
Open "C:\MiscData\Birthdays.txt" For Random As BdayFile
Input #PetsFile, strPetInfo 'чтение одного элемента из файла
Close PetsFile
Put BdayFile, 45, usrBDay 'запись одного элемента в файл
CloseBdayFile
```

Применяйте данный прием без опасения: вместо того чтобы собственноручно присваивать файлам номера, используйте функцию `FreeFile`, определяющую следующий доступный номер файла. Если значение, возвращаемое функцией, присвоить переменной, необходимость в запоминании номера просто отпадет. Вот пример применения данного приема: `Dim DiaryFile As Integer DiaryFile = FreeFile() Open «C:\MySecrets\Diary.txt» For Input As DiaryFile`

Закрытие открытых файлов

Будьте аккуратными — после того как работа с файлами завершена, их следует закрыть. Закрытие файла гарантирует, что все хранимые в памяти изменения действительно будут записаны на диск, а ресурсы, занимаемые файлами, освободятся и станут доступными для других приложений.

Для выполнения подобной задачи нужен, конечно же, оператор `Close`. Для закрытия определенного файла используется оператор вроде `Close #2` (или `Close PetsFile`, как в предыдущем примере). Для закрытия всех открытых файлов оператор `Close` используется сам по себе.

Чтение и запись данных

Есть несколько VBA-команд для записи и извлечения данных из файла. В приведенной ниже таблице описана их работа.

Пример команды	Использование	Пояснение
Оператор `Put`	Предназначен для записи переменных в файл	`Put #1, 1800, StrQuote` (записывает переменную `StrQuote` в двоичный файл, начиная с позиции `1800`); `Put #1, 15, usrCustomData` (записывает переменную `usrCustomData` в файле с произвольным доступом в запись 15)
Оператор `Get`	Предназначен для чтения из файла переменной, сохраненной с помощью оператора `Put`	`Gett #1, 1800, StrQuote` (читает строку данных из двоичного файла с позиции 1800 в переменную `StrQuote`); `Get #1, 15, usrCustomData` (читает определенные пользователем данные, хранимые в 15-й записи переменной `usrCustomData`)
Функция `Input`	Предназначена для чтения из файла данных, сохраненных с помощью оператора `Put`	`StrBigText = Input (400,#1)` (читает 4000 символов в переменную `StrBigText`)

Пример команды	Использование	Пояснение
Оператор `Write #`	Предназначен для записи списка перемены в последовательный файл, с созданием новой строки после каждого прохода	`Write #3, strShortString, strLongString, intLittleNumber` (записывает три переменные в строку последовательного файла)
Оператор `input #`	Предназначен для чтения списка из последовательного файла переменных, сохраненных с помощью оператора `Write #`	`Input #3, strShortString, strLongString, intLittleNumber` (читает данные для трех переменных из последовательного файла)

Глава 19

Еще о VBA-формах

В этой главе...

- Выбор цвета и добавление изображений к формам и элементам управления
- Настройка указателя мыши
- Использование дополнительных параметров элементов управления
- Проверка правильности вводимых данных и другие приемы

Материал настоящей главы дополняет сведения, изложенные в главе 10, предлагая вам советы о проектировании и программировании VBA-форм.

О внешнем виде форм и элементов управления

Хотя вы можете изменить параметры, относящиеся к внешнему виду форм и элементов управления, с помощью программного кода, чаще всего делать это не имеет смысла. Лучше воспользоваться диалоговым окном Properties для выбора всех необходимых настроек (подробности об использовании диалогового окна свойства редактора Visual Basic изложены в главе 10).

Выбор цветов

Значения свойств ForeColor и BackColor задают соответственно цвет переднего плана и цвет фона для форм и элементов управления. Для элементов управления значение свойства ForeColor задает цвет текста, размещаемого на элементе управления. Это свойство недоступно для тех элементов управления, на которых текст разместить нельзя. В случае форм значение свойства ForeColor задает цвет, который будет использоваться по умолчанию для всех новых размещаемых на форме элементов управления, а также для сетки на форме (эта сетка видна только в режиме проектирования формы).

Все относящиеся к выбору цветов свойства используются одинаково. После щелчка на кнопке со стрелкой в поле свойства, относящегося к цвету, появляется приятное на вид небольшое окно, в котором, несмотря на его небольшой размер, будет целые две вкладки (рис. 19.1).

На вкладке System (Системные) (рис. 19.1, слева) можно выбрать цвет из системной палитры Windows, т.е. из цветов, заданных в панели управления Windows для элементов интерфейса Windows. Если из списка выбрать пункт типа Desktop (Рабочий стол) или **Button Face** (Поверхность кнопки), VBA будет автоматически корректировать хранящееся в свойстве значение цвета при изменении соответствующих параметров в панели управления Windows.

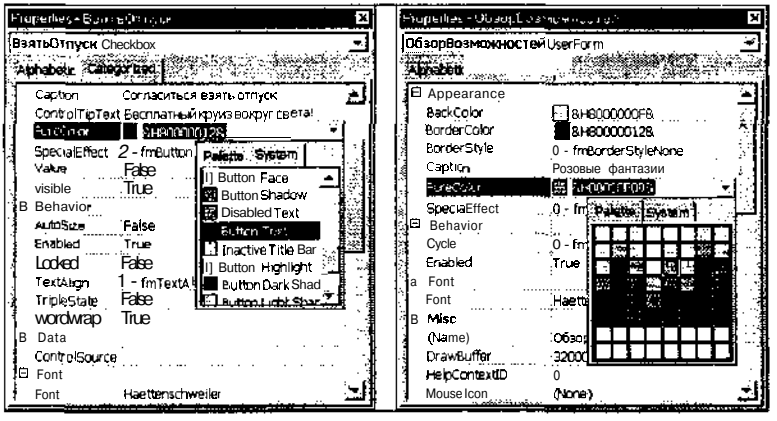


Рис. 19.1. Здесь показаны обе вкладки диалогового окна выбора цвета в VBA

На вкладке Palette (Палитра) (рис. 19.1, справа) можно выбрать цвет из предлагаемой палитры. VBA автоматически назначит свойству числовое значение выбранного цвета. В общем, довольно просто.

Выбор шрифтов

Хотя текст, который вы размещаете в форме и элементах управления, используя параметры, предлагаемые VBA по умолчанию, выглядит вполне сносно, можно выбрать гарнитуры шрифтов по своему вкусу. С помощью свойства Font (Шрифт), которое есть у форм и большинства элементов управления, вы можете выбрать для текста любой из шрифтов, установленных в системе.

Свойство Font формы на самом деле ничего не меняет в самой форме. Шрифт для заголовка формы, видимый в ее строке заголовка, задается системными установками в панели управления Windows. Свойство Font задает шрифт, который будет использоваться по умолчанию для текста тех элементов управления, которые добавляются в форму.

Из этого можно извлечь такой вывод: если вы собираетесь использовать какой-то нестандартный шрифт для всех элементов управления в форме, установите подходящее значение свойства Font для *формы*. Тогда вам не придется менять свойства Font для каждого из элементов управления по отдельности.

Чтобы выбрать шрифт для формы или элемента управления, найдите свойство Font в окне свойств, щелкните в соответствующем поле, а затем на появившейся в этом поле кнопке с многоточием. В результате на экране появится стандартное диалоговое окно Шрифт (рис. 19.2).

Если вы подумываете об использовании нестандартных шрифтов, имейте в виду два связанных с этим потенциально неприятных момента. Первый момент практический; если вы собираетесь распространять свою программу, то пользователям вашей программы нужно будет установить все использованные вами особые шрифты, поскольку в случае отсутствия каких-либо шрифтов Windows использует подстановки, и кто знает, как будут выглядеть шрифты, подставленные вместо ваших? Другой момент чисто эстетический: непривычные шрифты, как и комбинации различных гарнитур шрифтов, отвлекают внимание на себя, вместо того чтобы подчеркивать функциональное назначение формы.

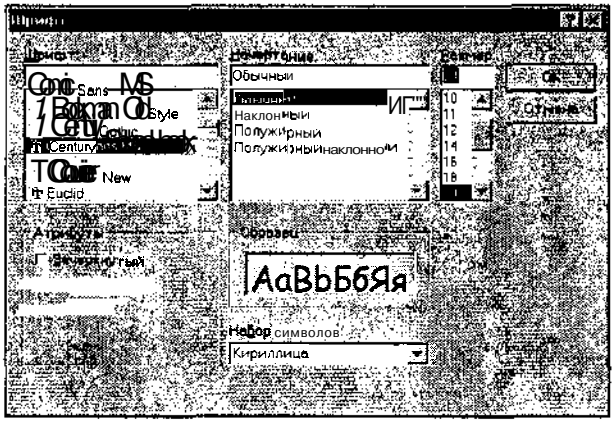


Рис. 19.2. В такомдиалоговомокневыбираетсяшрифт,которыйбудетиспользоватьсяпоумолчаниюдляформыилитекстанаэлементеуправления

Простые фокусы с мышью

В VBA есть несколько свойств, которые позволяют управлять тем, что увидят пользователи программы при разглядывании формы, двигая указатель мыши туда-сюда по экрану. Самое главное, что никакого программирования событий при этом не требуется!

Изменение вида указателя мыши

В любом профессионально изготовленном приложении вид указателя мыши зависит от того, над каким сегментом экрана он находится. И вы, как настоящий профессионал, непременно должны добавить такую эффектную возможность в свои VBA-программы. Тем более, для этого вам придется изменить значения лишь одного-двух свойств. Вот что нужно сделать.

1. **Выделите форму или элемент управления, для которого потребуется изменить вид указателя мыши.**
2. **Найдите в списке окна свойств свойство `MousePointer` и щелкните в поле этого свойства.**
3. **Выберите подходящий пункт из раскрывающегося списка этого поля.**

Все пункты раскрывающегося списка, кроме одного, соответствуют указателям мыши, задаваемым с помощью панели управления Windows (конкретно, на вкладке Указатели панели управления Мышь). Так что при выборе этих пунктов вид указателя в выполняемой программе будет на самом деле зависеть от той графики, которая назначена соответствующему типу указателя в панели управления Windows.

Если назначить специальный указатель мыши *форме,* то он получит соответствующий вид в рамках формы, включая все ее элементы управления (кроме тех, которым приписаны указатели иного вида).

Выбор изображения для указателя мыши

Не исключено, что вам понравится идея появления изображения черепа со скрещенными костями, когда указатель мыши будет пересекать недоступный элемент управле-

ния, — неважно, какой, и независимо от установок в панели управления Windows. Или, может быть, вы захотите видеть более приятное изображение, например золотую рыбку (рис. 19.3).

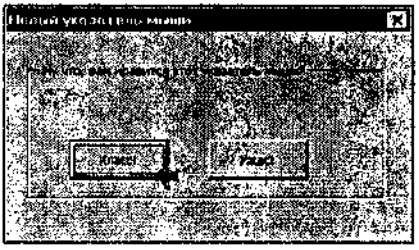


Рис. 19.3. Пользовательский указатель мыши на форме VBA

Превратить имеющееся графическое изображение в пользовательский указатель мыши несложно. Вот соответствующие инструкции.

1. **Создайте новую пиктограмму или возьмите готовую из коллекции рисунков.** Изображение нужно сохранить в стандартном формате Windows для пиктограмм, в файле с расширением `.ico`.
2. **Выполнив инструкцию, предложенную в** предыдущем **разделе "Изменение вида указателя мыши", установите** для **свойства** `MousePointer` **формы значение 99.**
3. **Щелкните в поле свойства** `MouseIcon`.
4. **Щелкните на кнопке, вызывающей диалоговое окно (на кнопке с многоточием).** VBA откроет диалоговое окно Load Picture (Загрузка рисунка), с помощью которого можно найти и открыть файл с картинкой для вашего указателя мыши.

Добавление всплывающих подсказок для элементов управления

Вы можете даже связать со своими формами всплывающие подсказки — небольшие текстовые сообщения, которые появляются рядом с указателем мыши при задержке последнего на секунду-другую на элементе управления. Всплывающие подсказки являются прекрасным средством ненавязчивого информирования пользователя о назначении элемента управления. И еще, всплывающие подсказки предусмотрены только для элементов управления, но не для содержащей их формы.

Чтобы назначить элементу управления всплывающую подсказку, просто напечатайте подходящий текст в поле свойства `ToolTip`. Пример того, что вы можете получить, показан на рис. 19.4.

Добавление изображений

Нельзя навсегда подавить свои художественные порывы, поэтому придет время, когда вы захотите украсить форму или ее элементы управления рисунками. В дополнение к возможности выбирать цвет для тривиальных текстовых окон, вы можете размещать изображения на кнопках, флажках, переключателях, фреймах и других элементах управления. Результат подобного творчества показан на рис. 19.5.

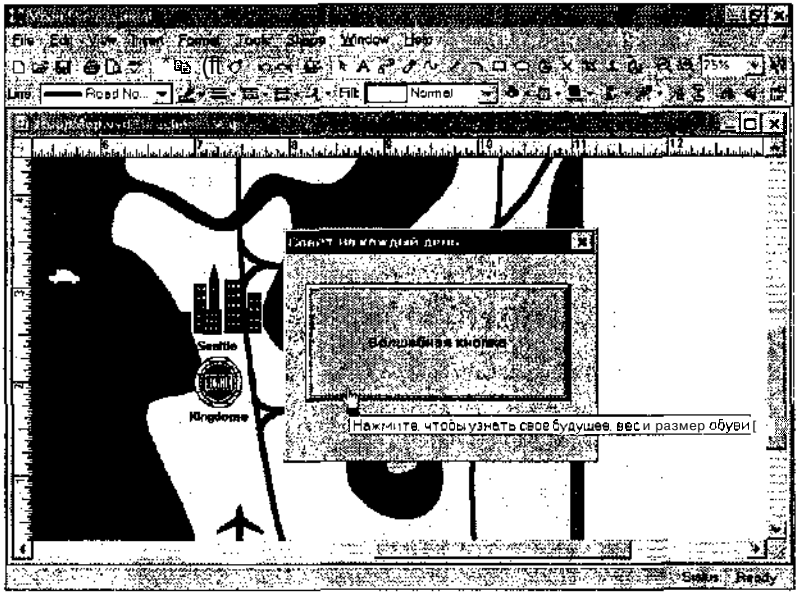


Рис. 19.4. ЧТО же делает эта кнопка? Указания на этот счет дает всплывающая подсказка

Добавление рисунков в формы и элементы управления

Элемент управления Image (Изображение) — тот элемент управления в панели Toolbox, пиктограмма которого напоминает картину,— можно использовать не только для того, чтобы размещать в нем изображения. Элемент управления Image — это, по сути, просто надпись без заголовка. Поэтому вы можете для размещения изображений использовать элемент управления надписью (Label), очистив поле свойства Caption (Заголовок), если текст на элементе управления не нужен.

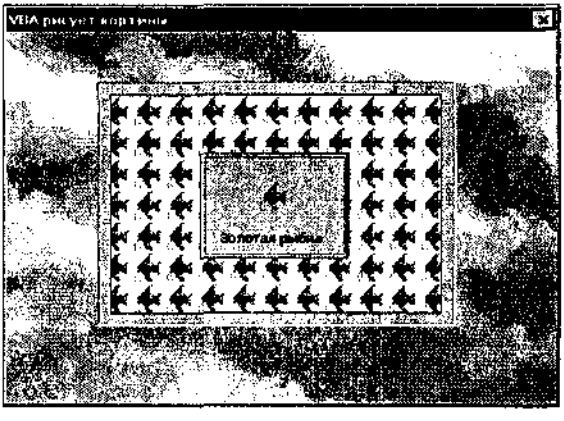


Рис. 19.5. В эту форму и ее элементы управления добавлены изображения

Не волнуйтесь, хотя работа с графикой не считается простой для начинающего программиста, но VBA превращает процесс в почти тривиальный. Вот инструкции, которым вам нужно следовать в данном случае.

1. **Щелкните на форме или элементе управления, куда нужно поместить прекрасный рисунок.**
2. **В окне свойств для выделенного объекта щелкните в поле свойства `Picture`.**
3. **Щелкните на открывающей диалоговое окно кнопке с многоточием, которая теперь появилась в поле свойства `Picture`.**
4. **В открывшемся диалоговом окне Load Picture (Загрузка рисунка) найдите файл с нужным изображением и щелкните на кнопке Открыть, чтобы добавить изображение в объект.**

   В форме (или элементе управления) появится рисунок.
5. **Используйте свойства `PictureAlignment`, `PictureSizeMode` и `PictureTiling` (если они есть у того объекта, с которым вы работаете), чтобы разместить рисунок так, как вам нравится. Вот некоторые подробности об этих свойствах.**

- ✓ Выбрав подходящее значение свойства `PictureAlignment`, можно разместить изображение либо по центру, либо в углу объекта.
- ✓ Выбрав подходящее значение свойства `PictureSizeMode`, можно либо растянуть изображение так, чтобы оно заняло всю поверхность объекта (соответствующее значение — `fmPictureSizeModeStretch`), либо пропорционально сжать изображение, чтобы оно поместилось в объекте (`fmPictureSizeModeZoom`), либо оставить размер изображения как есть, обрезав лишнее (`fmPictureSizeModeClip`).
- ✓ Для свойства `PictureTiling` допускаются всего два значения— `True` (т.е. VBA нужно размножить изображение и заполнить копиями весь объект) и `False`.

Удаление рисунков с помощью окна свойств

Поиграв некоторое время с рисунками в формах, не исключено, что вы придете к выводу, что "спартанский" вид для форм предпочтительнее. Чтобы удалить изображение из элемента управления или формы, щелкните в поле свойства `Picture` в окне свойств, а затем нажмите клавишу <Delete>. Другие методы не работают — ни нажатие <Ctrl+X>, ни нажатие клавиши пробела или печатание и даже нажатие клавиши <Backspace>.

Другие возможности форматирования

По большей части, другие опции меню Format применимы и к отдельным элементам управления, и к их группам. Выделив один или несколько элементов управления, вы можете использовать следующие опции.

- ✓ **Center in Form** (Центрировать в форме) — чтобы центрировать объекты по вертикали или по горизонтали. Если выделено несколько элементов управления, эта команда выставляет все элементы управления по средней линии, а не центрирует набор как одно целое (чтобы получить последнее, сначала нужно сгруппировать элементы управления).

- ✓ **Arrange Buttons** (Выстроить кнопки) — чтобы разместить одну или несколько выделенных кнопок у нижнего или у правого края формы. Команда Arrange **Buttons** работает и тогда, когда выделены не только кнопки, но перемещаются в результате выполнения этой команды только кнопки.
- ✓ Size **to Fit** (Задать размеры в соответствии с содержимым) — чтобы заставить VBA привести размеры элемента(ов) управления в точное соответствие с размерами содержащегося в нем текста. Эта команда действует одноразово — она не устанавливает для свойства `AutoSize` значение `True`, так что при изменении текста в элементе управления размеры последнего не изменятся.
- ✓ **Size to Grid** (Задать размеры по сетке) — чтобы сдвинуть все стороны элемента(ов) управления к ближайшим линиям сетки.

В редакторе Visual Basic нет команды, позволяющей равномерно распределить элементы управления в форме по горизонтали или вертикали. Однако приблизительно то же самое можно сделать достаточно просто с помощью такой последовательности действий {все упомянутые здесь команды находятся в меню **Format).**

1. **Разместите элементы управления с помощью мыши примерно так, как вам нужно.**
2. **Выделите эти элементы управления и активизируйте команду Align⇨Tops (Выровнять⇨По верху) или любую другую команду выравнивания по горизонтали.**
3. **Оставив все элементы управления выделенными, активизируйте команду Horizontal Spacing⇨Make Equal (Расстояние по горизонтали⇨Сделать одинаковым).**
4. **Сгруппируйте выделенные элементы управления (т.е. активизируйте команду Group (Сгруппировать)).**
5. **Активизируйте команду Center in Form Horizontally (Центрировать в форме по горизонтали).**

В вышеприведенной инструкции предполагалось равномерно распределить элементы управления по горизонтали. Чтобы распределить их равномерно по вертикали, выберите в пп. 2, 3 и 5 версии команд, соответствующие действиям по вертикали.

Дополнительно о работе с элементами управления

В главе 10 вы познакомились с наиболее важными элементами управления и их использованием. В настоящем разделе я дам вам дополнительные советы о работе с отдельными типами элементами управления, а также расскажу об элементах управления, которым не нашлось места в главе 10.

Советы об использовании текстовых полей

Как основные элементы для отображения и ввода данных, текстовые поля заслуживают особого внимания, поэтому я рассматриваю их в настоящем разделе.

Использование текстовых полей в качестве надписей

Как и надписи, текстовые поля могут содержать сообщения, которые вы адресуете пользователям. Однако обычно пользователь может изменять текст, отображаемый в текстовом поле. Если вы не хотите, чтобы пользователь изменял текст ваших сообщений, обязательно присвойте свойству `Locked` текстового поля значение `True`. Только имейте в виду, что пользователи все равно могут скопировать текст в буфер обмена, чего они не в состоянии сделать с текстом надписей. Вы можете предотвратить это, присвоив свойству `Enabled` значение `False`, однако в этом случае текст будет отображаться затененным.

Вы можете даже придать текстовому полю вид стандартной надписи, придав его свойству `BackColor` значение, соответствующее цвету форму, а свойству `SpecialEffect` — значение 0, чтобы сделать его "плоским".

Что такое секретный пароль?

Пароли используются для защиты важных данных от неавторизированного копирования, а также для того, чтобы создать у людей впечатление, что они действительно получат какие-то чрезвычайно важные сведения, если уж им предлагают ввести пароль. Какая бы из этих причин вам ни понравилась, создать текстовое поле VBA, предназначенное для ввода пароля, совсем несложно.

При выбранном текстовом поле в редакторе Visual Basic найдите свойство `PasswordChar` в диалоговом окне свойств, после чего введите подстановочный символ, который будет отображаться в текстовом поле (например, !) вместо символов, действительно вводимых пользователем. Текстовое поле все равно позволяет считывать введенные данные, но делать это может только программа (рис. 19.6).

Рис. 19.6. В этом текстовом поле вместо вводимых символов отображаются только восклицательные знаки

Конечно же, само по себе требование пароля не позволит защитить ваши данные. Сами данные и список паролей доступа должны быть зашифрованы, а ваша программа должна расшифровать данные в том случае, если пользователь ввел правильный пароль.

Использование многострочных текстовых полей

Для того чтобы данные в текстовом поле отображались корректно, текст понадобится разбить на несколько строк, поэтому вы должны присвоить свойству `MultiLine` значение `True`. В противном случае, даже если значение свойства `WrapText` равно `True`, весь текст будет размещен в одной строке, выходя за границы текстового поля.

Текст, который VBA для вашего удобства разделяет на несколько строк, на самом деле сохраняется в виде одной строки. Однако многострочные текстовые поля также позволяют пользователю начать "настоящую" новую строку с помощью, либо комбинации клавиш <Ctrl+Enter>, либо присвоения свойству `EnterKeyBehavior` значения `True` и нажатия клавиши <Enter>. Это позволит вам создать разрыв строки раньше, чем это автоматически сделает за вас VBA.

Если вы программист (а не пользователь программы), разместите текст в текстовом поле, после чего вы решайте, где ставить точки разрыва строк, Используйте любой из следующих двух методов.

- ✓ Если вы проектируете форму, щелкните на элементе управления два раза (но не дважды!), чтобы отобразить точку вставки. После этого воспользуйтесь комбинацией клавиш <Shift+Enter> для начала новой строки.
- ✓ В программном коде присвоите текст свойству Value элемента управления, используя строковые значения, объединенные с помощью символа возврата каретки.

На рис. 19.7 приведено несколько примеров использования свойств Wordwrap и MultiLine.

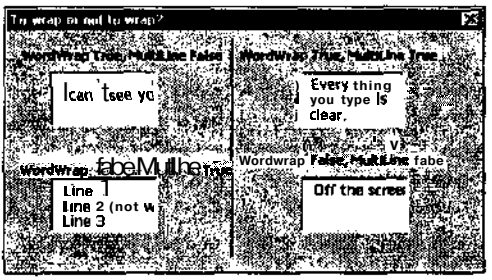


Рис. 19.7. В диалоговом окне вы видите свойства Wordwrap и MultiLine в действии

Добавление полос прокрутки

Какими бы ни были значения свойств MultiLine, Wordwrap или EnterKeyBahavior, в текстовом поле отображается не больше определенного количества текста. Хотя пользователь всегда может прокрутить содержимое текстового поля с помощью клавиш со стрелками, вам следует добавить полосы прокрутки к текстовому полю, присвоив свойству ScrollBars значение fmScrollBarsBoth. VBA достаточно сообразителен, чтобы отобразить полосы прокрутки только в том случае, если текстовое поле не в состоянии отобразить все содержимое сразу.

Создание форм с несколькими вкладками

Если вам необходимо работать с таким количеством элементов управления, что они просто не помещаются в одной форме, использование второй формы оказывается не самым лучшим решением проблемы. Вместо этого вы можете просто распределить все необходимые элементы управления по нескольким вкладкам формы.

Использование нескольких вкладок позволяет значительно "разгрузить" форму, что достаточно часто используется в диалоговых окнах различных Windows-приложений, как показано на рис. 19.8. Каждая вкладка, содержащая элементы управления, действует независимо от других. Элементы управления, которые вы добавляете на вкладку, связываются только с ней; вы сможете увидеть эти элементы управления только при отображенной соответствующей вкладке. Для отображения другой вкладки просто шелкните на ее "ярлыке" в верхней части формы.

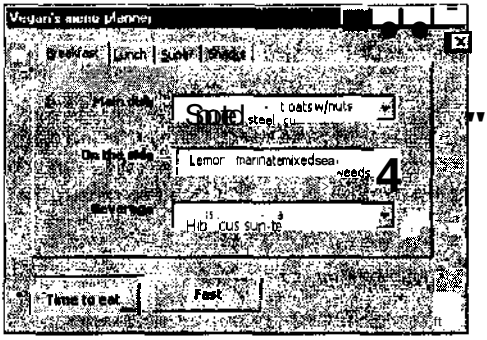


Рис.19.8. Форш, содержащая несколько вкладок с элементами управления

При проектировании отдельных вкладок формы выполняйте те же самые действия, что и при проектировании формы без вкладок: щелкните на нужном элементе управления в диалоговом окне Toolbox; затем перетащите принявший особый вид указатель мыши по диагонали, чтобы задать необходимый размер элемента управления. Форма со вкладками всегда содержит не меньше двух вкладок.

Работа с вкладками и свойствами их элементов управления

Форма, содержащая несколько вкладок, обладает определенными свойствами, точно так же, как и каждая ее отдельная вкладка. Если вам необходимо поработать с этими свойствами, вы должны твердо знать, что выбрали нужный элемент.

Свойства формы отображаются в диалоговом окне свойств сразу после того, как вы ее выделите. После этого, щелкнув на одной из вкладок, вы отображаете свойства именно ее. Для перехода к другой вкладке щелкните на ее ярлычке. Для вьщеления всей формы щелкните на ее строке заголовке или же воспользуйтесь раскрывающимся списком диалогового окна свойств.

Добавление элементов управления на вкладку

Для добавления кнопок, рамок и других элементов управления на вкладку формы просто перетащите их на соответствующую вкладку. Однако не забудьте перед тем, как перетаскивать элементы управления, выбрать соответствующую вкладку.

Добавление и удаление вкладок

Для добавления новой вкладки в форму выполните следующие действия.

1. Щелкните правой кнопкой на вкладках формы.
2. Выберите команду New Page из появившегося контекстного меню.

Удалить существующую вкладку также несложно, только удостоверьтесь в том, что вы удаляете нужную вкладку: редактор Visual Basic делает это немедленно, не задавая никаких вопросов о подтверждении подобных действий, а команда Undo в подобных ситуациях не работает. Для продолжения щелкните на вкладке правой кнопкой мыши и из появившегося контекстного меню выберите команду Delete Page.

Изменение заголовка отдельной вкладки

Для изменения заголовка вкладки измените подпись к ней. Это можно сделать, или введя новый текст в поле Caption диалогового окна свойств, или щелкнув правой кнопкой мыши на

ярлыке вкладки и выбрав из **появившегося** контекстного **меню** команду **Rename** для отображения соответствующего диалогового окна. В этом диалоговом окне вы сможете указать новый текст надписи.

Пусть вас не пугает слово **Rename** (Переименовать): **вы** действительно изменяете подпись к вкладке, а не ее имя (для изменения имени вкладки вам следует использовать диалоговое окно свойств). В любом случае, диалоговое окно Rename также позволит вам выбрать быструю клавишу и ввести текст подсказки. Вы можете изменить оба этих элемента с помощью диалогового окна свойств.

Изменение порядка следования вкладок

Для изменения порядка следования вкладок формы выберите команду **Move** из контекстного меню, отображаемого после щелчка правой кнопкой мыши на ярлычке вкладки. В диалоговом окне, показанном на рис. 19.9, щелкните на имени вкладки, расположение которой вы хотите изменить, после чего воспользуйтесь кнопками **Move Up** (Вверх) и **Move Down** (Вниз) для изменения расположения выбранной вкладки по отношению к другим.

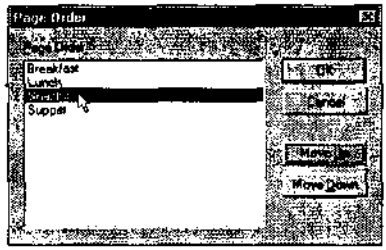


Рис. 19.9. Изменение порядка следования вкладок формы

Добавление специальных эффектов

Вы можете добавить действительно забавные эффекты, которые будут происходить при переходе от одной вкладки к другой. Вы можете заставить элементы управления постепенно появляться на вкладке от одного из ее краев, или постепенно проявляться новую вкладку на фоне старой. Для того чтобы увидеть все эти эффекты, вам следует выполнить форму.

Для добавления специальных эффектов для перехода между вкладками формы выполните следующие действия.

1. **Выберите необходимую вкладку.**
2. **В окне свойств выберите один из необходимых эффектов перехода из раскрывающегося списка `TransitionEffect`.**
3. **Задайте временной интервал (в миллисекундах) выполнения эффекта с помощью свойства `TransitionPeriod`.**

Если значение свойства `TransitionPeriod` равно 0, эффект отключен. Если же значение будет равно 500 (полсекунды), вы увидите эффект, но при этом задержек в работе программы не будет.

Совет о кнопках выбора

Обычно все кнопки выбора в определенной части формы (рамке, вкладке, или основной части формы) принадлежат одной группе. Однако, используя свойство GroupName, вы сможете определить несколько групп в одной и той же части формы. Вам необходимо только присвоить свойству GroupName одно и то же значение для определенных элементов управления.

Флажки

Для удобства работы пользователя вам следует визуально объединять наборы связанных флажков, чтобы пользователь всегда мог видеть, что определенные флажки относятся к одной группе. Для этого лучше всего подходят рамки.

Поскольку каждый флажок функционирует независимо от других, вам не стоит беспокоиться об их группировании каким-либо другим способом. Однако, если хотите, вы можете идентифицировать их как члены одной группы, указав одинаковое значение свойства `GroupName` для всех флажков одной группы. Это позволит вам намного проще определять принадлежность флажков при дальнейшем проектировании формы.

Выбор значений с помощью полос прокрутки и кнопок со стрелками

Когда вы увеличиваете уровень громкости звуковой системы или выключаете термостат нагревателя, вы используете настоящий элемент управления для выбора значения из диапазона доступных значений. В форме действие таких элементов имитируется с помощью полос прокрутки и кнопок со стрелками. Соответствующие примеры приведены на рис. 19.10.

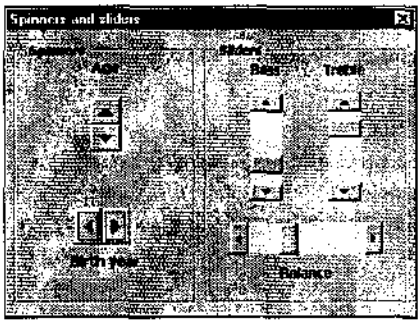


Рис. 19.10. В этом диалоговом окне полосы прокрутки и кнопки со стрелками используются для выбора различных значений

Конечно же, полосы прокрутки в среде Windows преимущественно используются для прокрутки видимой части документа или диалогового окна, если все представленные сведения не помещаются сразу. Но вы можете представить себе полосу прокрутки в более общем варианте: как элемент управления в виде бегунка, который позволяет прокручивать целый диапазон значений. Для выбора значения пользователь перетаскивает бегунок или щелкает на кнопке со стрелкой в ту или другую сторону.

Кнопки со стрелками чем-то напоминают полосы прокрутки, в которых собственно полоса и бегунок исчезли, а остались только кнопки. Подобные кнопки не позволяют прокручивать документ; они позволяют только выбирать значения.

Полосы прокрутки и кнопки со стрелками реагируют как на нажатие клавиш на клавиатуре, так и на щелчки мышью. Другими словами, вам не нужно программировать для них процедуры обработки событий для определения значений этих элементов управления, выбираемых в результате того или иного действия пользователя. Однако вам придется рассказать программе о том, что же необходимо делать с измененным значением.

Рисуем элементы управления

Вы размещаете полосы прокрутки и кнопки со стрелками в форме точно так же, как и любые другие элементы управления: щелкая на соответствующем значке в диалоговом окне **Toolbox,** после чего перетаскивая его на нужное место в форме. Однако при работе с полосами прокрутки и кнопками со стрелками необходимо помнить о некоторых нюансах.

И полосы прокрутки, и кнопки со стрелками можно ориентировать как по вертикали, так и по горизонтали. По умолчанию VBA определяет ориентацию элементов управления автоматически, основываясь на том, как именно вы перетаскиваете элемент управления. Если ширина элемента управления больше его высоты, ориентация горизонтальная. И наоборот, если ширина элемента управления меньше его высоты, ориентация вертикальная. VBA ориентирует элемент управления при каждом изменении этих параметров.

Если вы хотите получить широкую вертикальную полосу прокрутки, вы сможете строго задать необходимую ориентацию, воспользовавшись свойством `Orientation`.

Настройка элементов управления

После того как вы разместили полосу прокрутки или кнопки со стрелками в форме, вы должны выполнить две задачи, чтобы эти элементы управления работали должным образом.

- ✓ Вы должны указать диапазон значений, из которого пользователь сможет выбирать с помощью элемента управления.
- ✓ Вы должны обеспечить обратную визуальную связь с пользователем, чтобы он мог легко видеть, какое именно значение он выбрал в данный момент.

Выбор диапазона значений

Используйте свойства `Max` и `Min` полосы прокрутки или кнопок со стрелками для определения диапазона доступных значений. Эти свойства могут принимать только целые значения.

Имейте в виду, что несмотря на свои названия, свойства `Max` и `Min` относятся к расположению элемента управления, а не к числовым минимальным и максимальным значениям; значение свойства свойства `Max` может быть меньше значения свойства `Min`. Свойство `Min` относится к кнопке со стрелкой вниз. *(Я* настоятельно рекомендую вам поэкспериментировать с этими элементами управления.)

Организация обратной связи

Полосы прокрутки или кнопки со стрелками бесполезны до тех пор, пока пользователь не будет точно знать, какое же значение он выбрал. На рис. 19.11 приведено несколько примеров того, что пользователь видит, а также того, что он не должен видеть.

К сожалению, ни один элемент управления не создается со стандартным диапазоном значений. Для того чтобы связать значение, выбранное с помощью полосы прокрутки или кнопок со стрелками, с другим элементом управления, таким как текстовое поле, вам придется немного попрограммировать. Чаще всего мы обойдемся добавлением всего одной строки кода. Приведенный ниже фрагмент кода передает значение, полученное с помощью полосы прокрутки `sclWarpFactor` подписи `lblScrollBarReadout`, после щелчка на бегунке полосы прокрутки:

```
Private Sub sclWarpFactor_Click
    lblScrollBarReadout.Caption = sclWarpFactor.Value
End Sub
```

Обратите внимание, что для организации немедленной обратной записи, код, отображающий выбранное значение с помощью другого элемента управления, должен быть включен в процедуру обработки события `Click`, как и было показано выше.

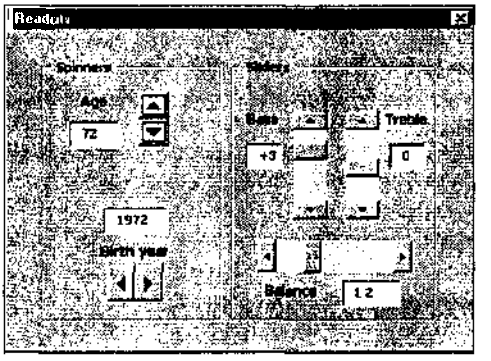


Рис. 15.11. Диалоговое окно, показанное на рис. 19.10, претерпело некоторые улучшения

Дополнительно о программировании форм

Программирование форм — это действительно высокое искусство. Очень сложно создать форму, которая будет вести себя так, как вы того ожидаете, не говоря уже о пользователе. Сведения, изложенные в настоящей главе, должны помочь вам лучше и быстрее достичь поставленной цели.

Использование переменных для ссылок на формы

Хотя в программном коде можно ссылаться на форму непосредственно по ее имени, не исключено, что вы предпочтете использовать для этого переменную. Например, чтобы уменьшить объем необходимого печатания, когда у формы длинное имя, или чтобы ускорить выполнение программы, когда в ней приходится отображать формы из разных проектов.

Поскольку формы являются объектами, к ним применима техника работы с объектами, рассмотренная в главе 10. Следующая процедура иллюстрирует процесс. Обратите внимание, что переменная должна объявляться как конкретная форма, а не как родовой объект UserForm:

```
Sub FormVariableDemo()

Dim frm1 As FormAnOpinion
Set frm1 = FormAnOpinion

' Изменение свойств и вызов методов с помощью переменной:
With frm1
     .Caption = "Все указанное выше"
     . Show
End With
End Sub
```

Строго говоря, оператор Set создает отдельную копию, или *экземпляр,* формы, присваивая его заданной вами переменной. Таким образом можно отображать несколько копий одной и той же формы, каждая из которых будет со своими собственными значениями в элементах управления. Для создания дополнительных экземпляров формы нужно использовать ключевое слово New. Вот как это выглядит:

```
' объявление переменных для форм
Dim frmOne As MultiForm
Dim frmTwo As MultiForm
' заполнение каждой переменной своим экземпляром формы
Set frmOne = MultiForm
Set frmTwo = New MultiForm ' здесь используется New
' отображение двух экземпляров формы
i'rmOne.Show
frmTwo.Show
```

Распознавание нажатий клавиш

Используйте события KeyPress, KeyDown и KeyUp, чтобы отвечать на нажатия клавиш пользователем. Событие KeyPress удобно использовать для распознавания клавиш с обычными "печатаемыми" символами (буквы, числа, знаки пунктуации), когда нужно обработать информацию, вводимую в текстовое поле или в поле со списком. С помощью этого события распознаются также многие из комбинаций типа <Ctrl+клавиша>, а также клавиша <Backspace>. Немного позже я покажу, как проверить или изменить напечатанный символ с помощью процедуры обработки события KeyPress.

События KeyDown и KeyUp, напротив, распознают практически любую посылаемую им комбинацию клавиш, включая выкрутасы типа <Alt+Shift+Ctrl+F9>. С этими событиями работать труднее, чем с KeyPress, но зато они позволяют использовать более широкий набор сочетаний клавиш. Например, можно создать процедуру обработки события KeyDown, которая позволяет с помощью комбинаций <Ctrl+←> и <Ctrl+→> соответственно уменьшать или увеличивать значение полосы прокрутки, например, на 10.

Дополнительные приемы проверки

В главе 10 мы обсудили основные приемы, позволяющие убедиться в том, что пользователь ввел в форму правильные сведения. Здесь мы обсудим еще несколько приемов.

Как убрать с экрана или изменить отдельные символы

В случае текстовых полей и полей со списком некоторые символы для вводимых данных не допускаются. Используйте процедуру обработки события KeyPress, чтобы убрать с экрана недопустимые символы в случае печатания их пользователем. Следующий программный код позволяет вводить только буквы и цифры:

```
Private Sub txtSerialNumber_KeyPress(ByVal KeyAscii _
          As MSForms.ReturnInteger)
' весь следующий блок является условием:
    If Chr(KeyAscii) < "0" Or _
    (Chr(KeyAscii) > "9" And Chr(KeyAscii) < "A") Or _
    (Chr(KeyAscii) > "Z" And Chr(KeyAscii) < "a") Or __
     Chr(KeyAscii) > "z" Then

    MsgBox "Недопустимый символ!"
    KeyAscii = 0 ' отбрасывание символа
End If
End Sub
```

Да, я признаю, что используемое здесь условие оказалось довольно длинным, но зато оно работает. Как только оператор If . . . Then обнаружит недопустимый символ, отображается

сообщение об этом. Затем идет оператор KeyAscii = 0; KeyAscii является аргументом процедуры обработки события KeyPress, поэтому он используется как локальная переменная в данной процедуре. Изменение его значения меняет код символа, передаваемого в текстовое поле. А поскольку само текстовое поле не воспринимает символ, имеющий код 0, напечатанный пользователем недопустимый символ исчезнет без следа.

Изменение значения аргумента KeyAscii позволяет изменять неправильно введенные данные на правильные. Например, следующую процедуру обработки события можно использовать для отображения и сохранения вводимого пользователем текста в виде текста, записанного буквами верхнего регистра:

```
Private Sub txtSerialNumber_KeyPress(ByVal KeyAscii _
        As MSForms.ReturnInteger)
KeyAscii = Asc(UCase(Chr(KeyAscii)))
End Sub
```

Оператор, осуществляющий преобразование, потребовал использования трех функций, вложенных одна в другую. Поскольку KeyAscii является числовым кодом символа, сначала приходится конвертировать этот код в строку с помощью функции Chr, затем перевести строку в верхний регистр с помощью UCase и, наконец, конвертировать строку снова в целое число с помощью Asc.

Отложенная проверка с помощью события BeforeUpdate

Если вы хотите перепроверять значение элемента управления при каждом изменении, создайте процедуру обработки события или используйте готовую из предыдущего подраздела. Но иногда предпочтительнее отложить проверку данных до момента, когда пользователь завершит ввод полностью. Некоторые будут признательны вам за возможность без посторонней помощи корректировать вводимые ими данные перед тем, как окончательно передать эти данные на проверку программе, — это они воспринимают как косвенное признание их умственных способностей.

Отложить проверку имеет смысл и в том случае, если она занимает много времени. Если придется сравнивать введенные данные с данными из какой-нибудь базы данных или данными из Internet, вы, наверное, не захотите заставлять пользователя ждать, пока программа будет искать данные для сравнения после каждого нажатия клавиши и каждого щелчка кнопки мыши.

Чтобы проверить значение элемента управления после того, как пользователь закончит ввод полностью, создайте процедуру обработки события BeforeUpdate. Эти события происходят, когда пользователь щелкает на другом элементе управления, нажимает клавишу <Tab> или нажимает комбинацию клавиш, назначенную другому элементу управления. VBA регистрирует событие BeforeUpdate непосредственно перед тем, как покинуть данный элемент управления, тут вы и можете отменить обновление данных и остаться на месте, предлагая пользователю исправить ошибку. Следующий пример показывает, как использовать оператор Cancel:

```
Private Sub txtSerialNumber_Change()
    If Len(txtSerialNumber.Value) > 5 Then
        MsgBox "Слишком много символов. Повторите ввод."
        Cancel
    End If
End Sub
```

Ожидание до закрытия формы

Иногда имеет смысл отложить проверку данных элемента управления до того момента, когда пользователь щелкнет на кнопке OK, закрывающей форму. Это приходится делать то-

гда, когда критерии проверки используют значения нескольких элементов управления в форме. Если вы выбрали такой тип проверки, поместите соответствующий программный код в процедуру обработки события Click кнопки OK. Предположим, вы разрабатываете форму, которая позволит пользователю ввести даты будущих важных событий. В форме есть список, в котором пользователь может указать, за какое время до наступления события нужно напомнить о нем. Если пользователь запланирует некоторую встречу на завтра, но попросит напомнить о ней за два дня до этого, вы можете сообщить о запрете ввода таких данных уже после того, как пользователь щелкнет на кнопке OK. Таким образом вам не придется просить пользователя ввести данные в элементы управления в определенном порядке, чтобы избежать появления сообщений об ошибке.

Часть V

Великолепные десятки

В этой части...

Эта короткая часть, оставленная на десерт, подогревает интерес к VBA множеством эффектных примеров. Попробовав представленное здесь, вы непременно захотите еще.

В главе 20 обсуждается ряд тонких приемов программирования, вполне способных заставить "летать" вашу VBA-программу. Среди рассмотренных тем — запись информации в реестр Windows и извлечение информации из него, использование возможностей других приложений и компонентов, чтение и запись дисковых файлов, а также добавление элементов управления ActiveX в панель элементов управления редактора Visual Basic. В главе 21 приведен обзор других ресурсов VBA, о которых вам полезно знать, — это журналы, Web-страницы, программные средства различных производителей.

Глава 20

Десятка (без трех) эффектных решений с помощью VBA

В этой главе...

- Сохранение установок программы в реестре Windows и извлечение их оттуда
- Доступ к объектам других приложений
- Работа с базами данных в VBA
- Сохранение информации на диске и ее чтение с диска
- Сохранение табличной информации в объектах Dictionary
- Создание своих собственных объектов
- Установка элементов управления ActiveX

Прочитав книгу до этой главы (а значит, большую ее часть), вы стали почти экспертом по VBA. Теперь вас уже не нужно постоянно вести за ручку, поэтому в этой главе, знакомящей с целым рядом важных приемов программирования в VBA, описания будут довольно сжатыми. Вы сможете воспользоваться предлагаемой здесь информацией на практике, чтобы почувствовать удовольствие от освоения достаточно тонких приемов программирования.

Сохранение информации в реестре Windows

Чтобы сохранить установки и значения других переменных, можно использовать небольшие файлы на диске, но реестр Windows как раз и предназначен для хранения подобных данных. VBA обеспечивает все необходимые средства для создания параметров реестра, размещения там данных и последующего извлечения информации.

Сохранение элемента информации в реестре осуществляет оператор SaveSettings, который автоматически создаст заданный вами параметр, если его в реестре не окажется. Синтаксис оператора следующий:

```
SaveSettings приложение, раздел, параметр, значение
```

Здесь приложение, раздел и параметр — это имена соответствующих уровней иерархии реестра. Все установки реестра для одной VBA-программы должны размещаться в одной, созданной специально для этой программы ветви реестра (используйте аргумент приложение для идентификации этой ветви). Сохраняемые в реестре данные можно разбивать на разделы и параметры совершенно произвольно, используя при этом произвольные имена.

Если вы назвали свою программу "Случайные цитаты" и хотите добавить в реестр параметры, хранящие информацию о том, сколько раз вызывалось диалоговое окно, содержащее случайную цитату, используйте для этого оператор типа:

```
SaveSettings "Случайные цитаты", "Параметры", _
    "Число вызовов", "6"
```

Последний аргумент, "6", задает реальные данные, которые должны быть помещены в реестр. На рис. 20.1 приведен результат выполнения этой строки программного кода.

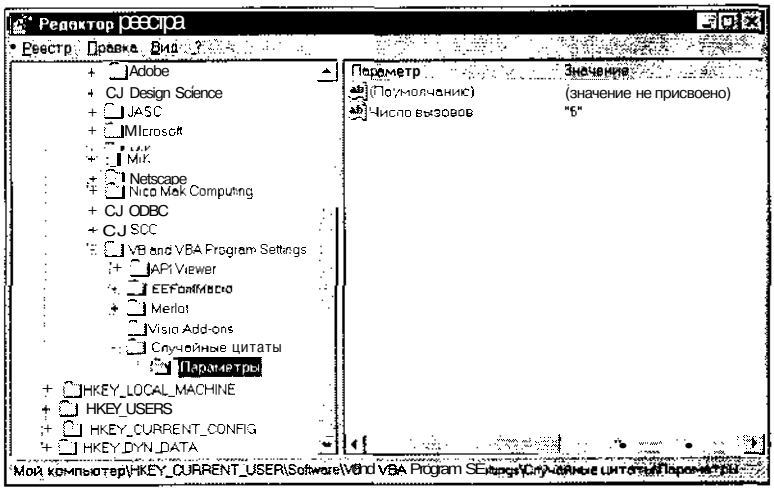


Рис. 20.1. Вид реестра Windows после добавления пользовательского параметра

Подобным образом для извлечения данных из реестра используется функция GetSetting. В ее синтаксисе, показанном ниже, первые три аргумента обязательны, а аргумент по_умолчанию не обязателен:

```
GetSetting приложение, раздел, параметр, по_умолчанию
```

Функция GetSetting используется, когда нужно выяснить, какое значение хранится в некотором параметре реестра. В программном коде нужно присвоить значение функции GetSetting переменной (вроде String или Variant), например, так:

```
sngЧисло = GetSetting("Случайные цитаты", "Параметры", _
    "Число вызовов")
```

Последний аргумент, аргумент по_умолчанию, определяет значение, которое должна вернуть функция GetSetting, если указанный параметр в реестре не найден.

В VBA есть еще две команды, предназначенные для работы с реестром.

- ✓ функция GetAllSettings возвращает список всех параметров и их значений в форме двумерного массива строк.
- ✓ Оператор DeleteSetting дает возможность удалить как значение, хранящееся в реестре, так и сам параметр.

Доступ к объектам других приложений

За рамками вашего VBA-приложения лежит ослепительная страна других приложений, и ничто не мешает вам отправиться туда в путешествие с помощью VBA. Все больше приложений инкорпорирует объектную модель COM, нагло выставляя свои объекты напоказ таким образом, что вы получаете возможность доступа к ним из своих программ.

В качестве простого примера предположим, что необходимо построить организационную диаграмму в Visio, основываясь на данных, хранящихся в папках Microsoft Outlook. Если вы уже знаете, как создавать элементы графики и связи между ними в Visio, новым для вас будет только создание программного кола, позволяющего извлечь информацию, "прописанную" в Outlook.

Программа Outlook не предлагает полномасштабную поддержку средств разработки VBA-программ (в частности, редактор Visual Basic), но Outlook выставляет для использования свои объекты в соответствии со стандартом COM. После установки связи с Outlook в VBA вы сможете использовать объекты Outlook в своей программе, как будто они родные для вашего "базового" приложения.

Основы межпрограммного взаимодействия

Первое правило очевидно; можно использовать объекты другого приложения, если оно установлено в системе. Если это условие выполнено, для работы с объектами приложения, поддерживающего стандарт COM, требуется выполнить ряд предварительных действий.

1. **В редакторе Visual Basic добавить ссылку на объектную библиотеку внешнего приложения.**
2. **Объявить переменные для объектов, которые придется использовать в программе.**
3. **Создать экземпляры соответствующих объектов с помощью функции `CreateObject`.**

В следующих трех подразделах эти шаги рассматриваются детальнее.

Добавление ссылки на внешнюю объектную модель

Чтобы информировать Visual Basic об объектной модели внешнего приложения, нужно добавить и активизировать *ссылку* на *объектную библиотеку* этого приложения. В окне редактора Visual Basic выберите Tools⇨References, чтобы открыть диалоговое окно **References** (Ссылки).

Если Outlook установлена на вашем компьютере, объектная библиотека Outlook уже должна присутствовать в списке **Available References** (Доступные ссылки) диалогового окна References. Ее нужно найти в этом списке и установить флажок рядом с ее именем, чтобы библиотека стала доступной. Если нужной библиотеки в списке не оказалось, добавьте ссылку в список сами, воспользовавшись кнопкой **Browse** (Обзор).

Объявление внешних объектов

Переменные для объектов из внешних приложений объявляются стандартным для VBA образом. Чтобы узнать, какие объекты внешнего приложения доступны, используйте либо обозреватель объектов редактора Visual Basic, либо файл справки этого внешнего приложения. Операторы следующего фрагмента программного кода объявляют переменные для объектов Outlook, которые предполагается позже использовать в программе:


```
Dim objOutlook As Outlook.Application
Dim objOLNamespace As Outlook.NameSpace
Dim colFolders As Outlook.Folders ' коллекция папок
Dim objPeopleFolder As Outlook.MAPIFolder
Dim colPeople As Outlook.Items ' коллекция контактов
Dim objPerson As Object ' один отдельный контакт
Dim strName As String
```

Создание внешнего объекта

Объявление объектов только говорит о вашем намерении использовать их. В дальнейшем вы должны создать необходимые объекты с помощью операторов Set внутри подходящих процедур.

При создании объекта из внешнего приложения используется функция CreateObject. Она открывает соответствующее приложение, которое и создает нужный объект. Возвращаемое функцией значение будет ссылкой на объект, которую можно присвоить подходящей переменной. А чтобы открыть внешнее приложение с уже готовым файлом или документом, используйте функцию GetObject.

Вот программный код процедуры из примера Outlook+Visio, создающей экземпляры нужных объектов:

```
Sub PeopleDiagram()
Set objOutlook = CreateObject("Outlook.Application")
' Используйте CreateObject("Outlook.Application.8")
' с другой версией Outlook
Set objOLNamespace = objOutlook.GetNamespace("MAPI")
Set colFolders = objOLNamespace.Folders ' все папки
Set objPeopleFolder = colFolders.Item("Personal Folders")
' переопределение переменных для движения внутрь папки
Set colFolders = objPeopleFolder.Folders
Set objPeopleFolder = colFolders.Item("Contacts")
Set colPeople = objPeopleFolder.Items
' вызов процедуры типа Sub, использующей данные Outlook
        ChartAName
End Sub
```

Обратите внимание, что во многих случаях — и в этом примере тоже — функция CreateObjecu нужна только один раз. После того как объект внешнего приложения создан, вы можете использовать его свойства и методы. Как правило, свойства включают другие объекты (в данном случае коллекция Folders, папки Personal Folders и Contacts, а также коллекцию объектов внутри папки Contacts).

Приложение, которое вы открываете из своего программного кода, выполняется в скрытом виде — на экране оно не отображается. Это очень удобно, если вы хотите использовать в своей программе данные другого приложения, не раздражая при этом пользователя. Но иногда все же нужно показать приложение в его обычной форме. Тогда, в зависимости от приложения, нужно либо установить для свойства Visible значение True, либо использовать метод Display, как здесь:

```
objPeopleFolder.Display
```

Использование внешних объектов

Теперь, наконец, вы готовы использовать объекты внешнего приложения в своей программе точно так же, как и те, которые пришли вместе с вашим VBA-приложением. Вот пример того, как это сделать в Visio при создании организационной диаграммы, использующей имена из папки Contacts, принадлежащей Outlook:

```
Sub ChartAName()
For Each objPerson In colPeople
```

```
        strName = objPerson.FullName
        ...(программный код построения диаграммы Visio на
                основе этих данных)
Next
End Sub
```

Программный код на Web-странице не содержит этой процедуры, поскольку я не знаю наверняка, что у вас есть Visio. Вместо вызова этой процедуры в процедуре `PeopleDiagram` просто отображается диалоговое окно при каждом обращении к данным Outlook. (Правда, у вас может не оказаться также и Outlook, в таком случае данный пример вообще не будет работать.)

Управление базами данных с помощью VBA

Если вы собираетесь использовать VBA для работы с данными, хранящимися в "реальных" базах данных типа файлов Microsoft Jet (формат баз данных, используемых Access), SQL Server или dBase, то вам, мой друг, определенно потребуется помощь. И придет она от объектной библиотеки. Объектная библиотека для работы с базами данных превращает базы данных и их компоненты — таблицы, запросы и отчеты — в настоящие объекты со свойствами и методами. А когда база данных представлена в виде множества объектов, вам уже не нужно копаться в деталях ее структуры. Более того, один и тот же набор объектов можно использовать для работы со многими базами данных различных типов. Подробные сведения об управлении базами данных с помощью VBA изложены в главе 17.

Работа с файлами

С помощью специальной внешней объектной библиотеки VBA позволяет использовать объектно-ориентированный подход при работе с дисковыми файлами, в частности для чтения содержимого каталогов и копирования файлов. Соответствующие приемы программирования обсуждаются ниже, в разделах "Работа со свойствами файлов" и "Копирование, изменение и удаление файлов". Можно использовать также функции и операторы VBA, краткое описание которых вы найдете в табл. 11.9 (см. раздел "Работа с файлами").

Наступит время, когда вы захотите, чтобы ваша VBA-программа сохраняла информацию в файле на диске. Может быть, вам потребуется сохранить выбранные пользователем параметры установки, а может, вы пожелаете сохранить значения некоторых переменных, чтобы использовать их при следующем запуске программы. Или программе нужно сохранить в виде файла результаты множества вычислений, или работать с информацией, хранящейся в текстовом документе.

Независимо от того, доступ к какому объему информации вам нужен, VBA делает работу с файлами достаточно простой. Вам не придется выяснять, каким образом организована работа с файлами в Windows, — несколько простых объектов VBA и их методы сделают практически все за вас.

Учитывая меньшую значимость, я не говорю здесь о чтении и записи файлов документов вашего "базового" VBA-приложения. Ясно, что VBA-программы могут работать с информацией, содержащейся в таких документах, и обращаться с файлами документов как с единым целым (обычно с помощью методов Open и Save).

Принципы работы с файлами в VBA

Как уже говорилось в главе 11, VBA позволяет работать с файлами как с объектами и манипулировать файловыми объектами с помощью их свойств и методов, Но это не стало органической частью VBA, а обеспечивается некоторой внешней объектной библиотекой. Как объясняется в следующем разделе, перед работой с файлами как с объектами необходимо добавить в программу ссылку на эту библиотеку.

VBA имеет также встроенную схему обработки файлов, в рамках которой файлы *не* трактуются как объекты. В этом случае для выполнения файловых операций используются операторы и функции, а не свойства и методы. Эта схема в VBA до сих пор работоспособна, но для большинства задач предпочтительнее иметь дело с файловыми объектами. Список соответствующих операторов и функций приведен в табл. 11.9, а информация об их использовании изложена в главе 18.

Ссылки на библиотеку Microsoft Scripting Runtime

Основные средства, необходимые для работы с файлами как с объектами, обеспечиваются объектной библиотекой Microsoft Scripting Runtime. Прежде чем создавать относящийся к файлам программный код, в VBA-проект нужно добавить ссылку на эту библиотеку. А перед тем, как сделать это, нужно убедиться в том, что соответствующая библиотека установлена на компьютере.

Библиотека Microsoft Scripting Runtime содержится в файле с именем `SCRRUN.DLL`, размещенном в папке `Windows\System`. Библиотека устанавливается автоматически при инсталляции Windows 98 или Windows NT с дополнением Option Pack, а также если в систему устанавливается VBA 6. Если вы работаете с VBA 5 в Windows 95 или если по каким-то другим причинам библиотеки Microsoft Scripting Runtime в вашей системе не оказалось, загрузите ее на свой компьютер с Web-страницы *Microsoft* для разработчиков (`msdn.microsoft.com/scripting/`).

Убедившись, что в вашей системе есть библиотека Microsoft Scripting Runtime, добавьте ссылку на нее в каждый из проектов, где предполагается ее использовать. Для этого выполните следующее.

1. Выберите проект в окне проводника проектов.
2. Выберите **Tools⇨References,** чтобы открыть диалоговое окно **References (Ссылки).**
3. В списке диалогового окна найдите пункт **Microsoft Scripting Runtime** и установите флажок, соответствующий этому пункту.
4. Закройте диалоговое окно.
   Трам-та-ра-рам-тарам! Все готово.

Эти действия нужно повторить для каждого проекта, работающего в библиотеке Microsoft Scripting Runtime, независимо от используемой версии VBA (о внешних объектных библиотеках см. выше, в разделе "Доступ к объектам других приложений").

VBA-программы, использующие библиотеку Microsoft Scripting Runtime, можно поставлять отдельно только тем пользователям, у которых в системе установлен соответствующий файл . DLL. В противном случае вместе с программой нужно поставлять и файл . DLL (с инструкциями по его установке).

Доступ к файлам

Итак, библиотека Microsoft Scripting Runtime установлена, соответствующая ссылка в VBA-проект добавлена. Теперь вы готовы приступить к созданию объектно-ориентированного программного кода, обеспечивающего возможность манипуляций с любыми файлами на диске. При работе с файлом как с объектом выполняются следующие действия.

1. **Создание `FileSystemObject` — объекта высшего уровня, обеспечивающего доступ к файлам на диске.**
2. **Вызов подходящего метода объекта `FileSystemObject` для открытия нужного файла или для создания нового файла для использования.**
3. **Работа с файловым объектом с помощью его методов и свойств.**

Практическая реализация этих действий иллюстрируется следующим фрагментом программного кода. В нем сначала создается объект `File` для файла `lacewings.txt`, а затем этот файл копируется в другое место на диске:

```
Dim objFileSystem As FileSystemObject
Dim objTextStream1 As File
Set objFileSystem = _
    CreateObject("Scripting.FileSystemObject")
Set objFile1 = _
    objFileSystem.GetFile("C:\bugs\lacewings.txt")
objFile1.Copy ("C:\Мои документы\buggy.txt")
```

Работа со свойствами файлов

Имея для работы объект `File`, несложно получить доступ к содержащейся в файле информации с помощью свойств объекта. Например, с помощью свойства `Size` можно выяснить размер файла

```
Dim lngFileSize As Long
lngFileSize = objFile1.Size
```

или с помощью свойства `DateLastModified` узнать, когда файл изменялся последний раз:

```
Dim dateFileDate As Date
dateFileDate = objFile1.DateLastModified
```

Копирование, изменение и удаление файлов

С помощью подходящих методов объект `File` можно скопировать, переместить или удалить:

```
objFile1.Copy "c:\героические усилия\"
objFile1.Move "c:\минимальные результаты\"
objFile1.Delete
```

Заметьте, что если вы не меняете имя файла, в операторах, использующих методы `Copy` и `Move`, нет необходимости указывать имя, а нужно указать только путь назначения. (Не забудьте в конце имени пути добавить обратную косую черту.) И кстати, объект `File` сохраняет ассоциацию с перемещенным файлом.

Чтобы копировать, перемещать или удалять *группы* файлов, используйте методы `CopyFile`, `MoveFile` и `DeleteFile` объекта `FileSystemObject`, а не индивидуальный объект `File`. В спецификациях имен файлов для методов объекта `FileSystemObject` можно использовать символы подстановки. Объект `FileSystemObject` имеет еще методы `CopyFolder`, `MoveFolder` и `DeleteFolder`, которые применимы к папкам вместе с вложенными в них структурами.

Чтение и запись данных

Кроме копирования, перемещения и удаления файлов, вас, возможно, заинтересуют и данные, хранящиеся внутри файлов. В этом разделе обсуждается объектно-ориентированный подход к организации чтения и записи информации.

Объекты, относящиеся к файлам, можно использовать для организации доступа к содержимому файла только как к непрерывному блоку текста. Конечно, этот текст может содержать и числовые данные, ведь ничто не заставляет вас хранить и читать только имена, прозу и поэзию. Но доступ к файлу может быть только последовательным — нельзя произвольно перескакивать с одного места в файле на другое.

Объекты `File` не обеспечивают средства для работы со структурированными данными, как в простых базах данных. В структурированных файлах каждая запись занимает заранее определенное пространство, что позволяет считывать и записывать записи просто по их номерам. В VBA можно обеспечить и такой случайный метод доступа, но не с помощью объектов, а используя операторы VBA, такие как Open, Put и Get.

Открытие объектов текстовых потоков

Чтобы читать данные из файла или записывать их в файл, сначала нужно открыть файл как *текстовый поток*. В программном коде работа с текстовым потоком означает работу с объектом `TextStream`. Начать эту работу можно одним из следующих трех способов.

- ✓ Открыть существующий объект `File` в режиме текстового потока.
- ✓ Создать новый файл и одновременно открыть его как текстовый поток.
- ✓ Открыть существующий на диске файл как текстовый поток.

В следующем примере процедуры этими тремя способами в указанном порядке открываются три отдельных объекта `TextStream`, затем в один из этих объектов записываются данные, и объекты закрываются:

```
Sub TextStreamDemo()

Dim objFileSystem As FileSystemObject
Dim objFile1 As File
Dim objTextStream1 As TextStream
Dim objTextStream2 As TextStream
Dim objTextStream3 As TextStream

Set ObjFileSystem = _
     CreateObject("Scripting.FileSystemObject")

' Создание объекта File и использование его для создания
объекта текстового потока
Set objFile1 = _
     objFileSystem.GetFile("C:\СуществующийФайл.txt")
Set objTextStream1 = _
     obj File1.OpenAsTextStream(ForReading)

' Создание файла и открытие его как объекта _
' текстового потока
Set objTextStream2 = _
```

```
        objFileSystem.CreateTextFile("C:\НовыйФайл.txt")

        ' Открытие существующего файла как _
' текстового потока
        Set objTextStream3 = _
    objFileSystem.OpenTextFile("C:\СтарыйФайл.txt")

' Запись двух строк текста в один из потоков
objTextStream2.WriteLine _
"Взвейтесь кострами, синие ночи!"
objTextStream2.WriteLine "Мы - пионеры, дети рабочих."

' Закрытие всех трех текстовых потоков
objTextStream1.Close
objTextStream2.Close
objTextStream3.Close

End Sub
```

Хотя во всех трех случаях создаются объекты TextStream, они не взаимозаменяемы. От выбранного в каждом конкретном случае способа зависит и способ доступа к данным в соответствующем файле. Подробнее об этом говорится в следующем разделе.

Выбор и использование различных режимов доступа к данным

Каждый конкретный объект TextStream позволяет либо читать данные из файла, либо записывать данные в файл, либо и то, и другое. Объект TextStream позволяет выбрать для него один или несколько из *трех режимов* ввода-вывода. Один из этих режимов только для чтения, и есть два разных режима для записи — обычная запись (когда предполагается запись символов с самого начала текстового потока) и добавление (когда предполагается добавление символов в конец текстового потока).

Доступные для каждого конкретного объекта TextStream режимы зависят от того, каким способом создан текстовый поток, а в случае метода OpenAsTextStream объекта File — от указанного для него режима. Вот соответствующая сводка.

Метод	Доступные режимы ввода-вывода для определенного объекта текстового потока
CreateTextFile (объект FileSystemObject)	Чтение и запись
OpenTextFile (объект ileSystemObject)	Чтение и добавление
OpenAsTextStream (объект File)	Чтение, запись или добавление (только один из режимов)

Когда объект TextStream создается из объекта File, режим ввода-вывода задается как аргумент метода OpenAsTextStream с помощью трех соответствующим образом названных констант: ForReading (для чтения), ForWriting (для записи) и ForAppend-

ing (для добавления). Например, если нужно добавить текст в конец существующего файла, откройте текстовый поток с помощью следующего оператора:

```
Set objTS = _
    objFile.OpenAsTextStream(ForAppending)
```

Чтение и запись файловых данных

Открыв объект TextStream в подходящем режиме доступа, можно начинать работу с содержимым файла. В вашем распоряжении для этого есть несколько простых методов.

Можно читать и записывать данные как отдельные символы или строки. Если только все элементы данных не равны по длине, я рекомендую построчное считывание и построчную запись. Когда читаются строки данных, вам не нужно знать длину строк, а только их номер. Сама же "строка" может состоять из любого числа символов и даже вообще не содержать ни одного символа.

В любом случае, чтобы читать данные из текстового потока, присвойте переменной значение, возвращаемое одним из подходящих методов объекта TextStream. Следующий фрагмент программного кода подскажет вам, как это сделать:

```
' чтение следующей строки из файла в переменную
strНекоторыйТекст = TextStream.ReadLine
```

Для чтения предусмотрены следующие методы.

Метод (объекта TextStream)	**Выполняемое действие**
Read (*число_символов*)	Чтение заданного числа символов начиная с текущей позиции в файле
Метод (объекта TextStream}	**Выполняемое действие**
ReadLine	Чтение всех символов начиная с текущей позиции в файле до следующего символа перехода на новую строку
ReadAll	Чтение всего файла
Skip(*число_символов*)	Пропуск заданного числа символов при чтении файла. Следующее чтение должно начаться с первого из символов, следующих за пропущенными
SkipLine	Пропуск следующей строки в файле. Следующее чтение должно начаться с первого из символов, следующих за пропущенной строкой

Чтобы записать данные в текстовый поток, задайте текст для записи в виде строки как аргумент метода Write или WriteLine, как в следующем примере:

```
' запись строки
objTextStream.Write "Алло, центральная!"
' запись значения переменной с последующим символом
' перехода на новую строку
objTextStream.WriteLine strMyTwoBits
```

Метод (объекта TextStream)	Выполняемое действие
Write(*строка*)	Запись в текстовый поток указанной строки
WriteLine(*строка*)	Запись в текстовый поток указанной строки с последующим символом перехода на новую строку
WriteBlankLines(*число_строк*)	Запись в текстовый поток указанного числа символов перехода на новую строку

Закрытие текстовых потоков

Завершив чтение текстового потока или запись в текстовый поток, всегда закрывайте объект TextStream, чтобы освободить системные ресурсы и память и чтобы другие программы и пользователи смогли получить доступ к соответствующему файлу. Здесь нужен метод Close:

```
objTextStream.Close
```

Ограничения при работе с текстовым потоком

Текстовые потоки хороши для чтения и записи относительно небольших по объему порций данных. Основная проблема состоит в том, что нельзя свободно перемещаться внутри текстового потока. Методы чтения начинают чтение данных с самого начала файла и читают его до конца. И хотя при чтении можно использовать методы Skip и SkipLine, чтобы перепрыгнуть заданное число символов или строк, в обратном направлении перескакивать нельзя — выполнение оператора типа objTextStream.Skip (-10) порождает ошибку.

К сожалению, объекты TextStream не имеют методов или свойств, позволяющих прямо задать стартовую позицию для следующего чтения. Однако примерно то же самое можно сделать по-другому. Сначала нужно закрыть и снова открыть текстовый поток, а затем с помощью метода Skip пропустить на одну позицию меньше, чем номер нужной позиции. Например, если вы хотите прочитать 25 символов, начиная с 5-го символа в файле, соответствующую задачу выполнят следующие операторы:

```
objTextStream.Close
Set objTextStream = bjFile.OpenAsTextStream(ForWriting)
objTextStream.Skip(4)
objTextStream.Read(25)
```

При записи данных в текстовые потоки вы еще больше ограничены, чем при чтении, поскольку вообще не допускаются пропуски в файле. Каждая последующая операция записи начинается там, где закончилась предыдущая, и при этом переписывается любой текст, который может оказаться на соответствующих позициях в файле.

Поэтому, если вам нужно внести в каком-то месте файла некоторые изменения, не затрагивая при этом остальной текст, прочитайте все содержимое файла в строку, соответствующим образом измените ее, а затем замените все данные в файле новыми данными.

Использование объектов Dictionary

В объектах Dictionary (Словари) удобно хранить табличную информацию, организованную в два столбца. В каждой строке такой таблицы в одном столбце хранится *имя* элемента, а в другом — нужное вам значение. Действительно, очень похоже на словарь; заголовки

словарных статей — это имена элементов, а словарные определения — значения. Правда, объекты Dictionary допускают хранение значений любых типов.

С объектами Dictionary можно работать и в VBA 5, и в VBA 6. Но эти объекты не встроены в VBA. Они находятся в библиотеке Microsoft Scripting Runtime, использование которой уже обсуждалось в разделе "Работа с файлами". Чтобы использовать объекты Dictionary в VBA-программе, нужно выполнить инструкции, приведенные там под заголовком "Ссылки на библиотеку Microsoft Scripting Runtime".

Базисные сведения об объектах Dictionary

При объявлении переменных для словарей и последующем их создании используйте приемы, описанные выше в разделе "Доступ к объектам других приложений". Например:

```
Dim dictBigCats As Scripting.Dictionary
Set dictBigCats = CreateObject("Scripting.Dictionary")
```

Теперь можно добавлять в словарь имена элементов и значения с помощью метода Add:

```
dictBigCats.Add "Гепард", "Быстрый и поджарый"
dictBigCats.Add "Лев", "Рычащий и лохматый"
dictBigCats.Add "Пантера", "Черная, а не розовая"
```

Лучше, чем коллекции

Объекты Dictionary в некоторых довольно важных деталях совершеннее подобных им объектов Collection, которые обсуждались в главе 12, по следующим причинам.

- ✓ Не требуется знать имена элементов для доступа к ним и к соответствующим им значениям, поскольку возможен доступ ко всем элементам *сразу* с помощью цикла For Each . . . Next. Например:

```
For Each airplane in dictBiplanes
  MsgBox dictBiplanes(airplane)
Next
```

- ✓ Можно изменить значение элемента, используя имя элемента, просто присвоив новое значение:

```
dictVacationDestinations("Number1") = "Fresno"
```

- ✓ В коллекциях приходится сначала удалять исходный элемент, а затем добавлять новый.
- ✓ Можно удалить все элементы словаря с помощью метода RemoveAll. Для выполнения той же задачи в коллекции приходится использовать цикл For Each . . . Next.

За неимением места здесь не обсуждаются приемы сортировки элементов словарей и выбора подмножеств элементов, удовлетворяющих заданным критериям. Замечу лишь, что такие типичные для баз данных функции реализовать со словарями достаточно просто. Подробные инструкции и примеры программного кода вы найдете в статье Bruce McKinney, *Understanding the Dictionary Class, Visual Basic Programmer's Journal,* July 1999, (издательство Fawcette Technical Publications), доступной также через Internet по адресу www.vbpj.com.

Пользовательские объекты

Освоив использование встроенных объектов VBA и других объектных библиотек, вы, наверное, захотите создать свои собственные объекты. Хотя можно достичь немалого и с по-

мощью обычных процедур типа Sub и Function, выделение части программного кода в виде объектов обладает реальными преимуществами.

- ✓ Размещение всего программного кода, обрабатывающего некоторое множество данных, внутри отдельного объекта уменьшает вероятность внесения ошибок при модификации программы.
- ✓ Программа будет легче для чтения и понимания.
- ✓ Можно создать сколько угодно копий объекта, причем для создания каждой копии понадобится всего пара коротких операторов.
- ✓ Упоминание преимуществ *полиморфизма,* вероятно, потребует дополнительных разъяснений, но... использование одних и тех же свойств и методов с разными классами объектов все же оказывается удобным. А именно это подразумевает полиморфизм. К сожалению, я не могу тут развивать эту тему, но вы должны знать, что существует такая мощная техника программирования. Подробно полиморфизм обсуждается в книгах, цель которых — более глубокое изучение VBA.

Как вы уже знаете (если прочитали главу 12), объект состоит из данных (свойств объекта) и программного кода, изменяющего эти данные (методов объекта). Поскольку свойства являются просто переменными, а методы — процедурами, создание программного кода, задающего объект, не такая уж сложная задача. Но при этом необходимо следовать определенным правилам, чтобы VBA мог распознать, что именно вы собираетесь сделать. Следующие несколько разделов посвящены описанию этих правил.

Создание модулей классов

В VBA *класс* является шаблоном, по которому создаются подобные объекты. Класс определяет, какие свойства, методы и события должен иметь объект и как должен "вести" себя каждый из этих компонентов.

Чтобы создать класс, начните со вставки нового модуля класса в VBA-проект (для чего выберите Insert⇨Class Module). Окно модуля класса выглядит и работает точно так же, как и обычное окно программного кода. Прежде чем двигаться дальше, задайте имя нового класса в строке (Name) в окне свойств.

Компоненты определения класса

Типичный класс имеет три главных компонента.

- ✓ Объявленные локальными переменные, предназначенные для использования внутри объекта.
- ✓ Открытые процедуры свойств, позволяющие процедурам из стандартных модулей прочитать или изменить текущие значения свойств.
- ✓ Открытые процедуры методов, задающие действия, выполняемые методами объекта.

Представленное ниже определение простого класса Thermostat включает все три этих компонента. Сам по себе пример не предназначен для выполнения каких-либо полезных задач, но он работает и есть на Web-странице на случай, если вы захотите его испытать. Если же вы предпочтете набрать программный код самостоятельно, с помощью окна свойств назовите модуль класса Thermostat, а затем введите в окне его программного кода следующее:

```
Private sngDegrees As Single ' переменная свойства
' Программный код процедуры свойства Let Temperature:
Public Property Let Temperature(ByVal sngInput As _
          Single)
     sngDegrees = sngInput
End Property
' Программный код процедуры свойства Get Temperature:
Public Property Get Temperature() As Single
     Temperature = sngDegrees
End Property
' Программный код процедуры метода CalculateEnergyUse:
Public Sub CalculateEnergyUse()
     Const cstConversionFactor =2.45
     Dim dblResult
     dblResult = sngDegrees * 365 * cstConversionFactor
     MsgBox "Годовой расход энергии на поддержание " & _
          "установок термостата оценивается в " & _
          dblResult & " Ватт."
End Sub
```

Объявлениепеременныхкласса

Для объявления переменных, которые понадобятся в нескольких свойствах или методах, используйте раздел Declarations в самом начале модуля класса. Всегда объявляйте эти переменные как локальные (Private), ведь объекты в основном предназначены для того, чтобы запретить программе получить прямой доступ к данным. Переменные, которые будут использоваться только в одном свойстве или методе, должны там и объявляться.

Для каждого из свойств объекта нужно объявить как минимум по одной переменной. Имя переменной *не* должно совпадать с именем свойства (чуть позже я объясню, как задать имя свойства). Можно объявить и другие данные, которые объект будет использовать внутри себя и которые будут недоступными для других частей вашей программы.

Созданиепроцедурсвойств

Секрет наделения объекта свойством состоит в написании пары специальных процедур — *процедур свойств* Property Let и Property Get. Обе процедуры в паре должны иметь одинаковые имена.

Именем свойства будет то, которое вы выберете для процедур Property Let и Property Get. Ясно, что оно должно описывать содержимое или функцию свойства.

Да, и еще: если создаваемое свойство будет представлять ссылку на другой объект, вместо процедуры Property Let в вышеуказанной паре процедур используйте процедуру Property Set. В остальном такое свойство ничем не отличается от свойств, использующих другие типы данных.

Установкасвойствобъектаспомощьюпроцедур Property Let

Процедура Property Let устанавливает значение свойства. В своей простейшей форме процедура Property Let берет переданное ей в виде аргумента значение и присваивает его переменной, представляющей свойство. В предыдущем примере это выглядело так:

```
Public Property Let Temperature(ByVal sngInput As Single)
     sngDegrees = sngInput
End Property
```

Когда в основной части программы выполняется оператор, устанавливающий свойство, например

```
Thermostat.Temperature = 75
```

VBA вызывает процедуру Let Temperature со значением 75 для ее аргумента.

Ясно, что процедуры свойств не обязаны содержать лишь по одной строке программного кода. Можно добавить и другие операторы, например, проверяющие введенное значение на допустимость прежде, чем присвоить его свойству, или выполняющие какие-то иные действия.

Чтение свойств объекта с помощью процедур Property Get

Процедура Property Get подобна процедуре типа Function в том смысле, что она возвращает значение — конечно же, значение свойства, Как и в случае процедуры типа Function, значение, которое должно быть возвращено, присваивается имени процедуры, которое в данном случае оказывается именем свойства. Вот опять фрагмент предыдущего примера:

```
Public Property Get Temperature() As Single
    Temperature = sngDegrees
End Property
```

Другие части вашей программы могут вызывать процедуру Get Temperature, чтобы присвоить возвращаемое этой процедурой значение переменной либо использовать это возвращаемое значение в условных операторах, как в следующем примере:

```
sngCurrentSetting = Thermostat.Temperature

If Thermostat.Temperature > 80 Then
    MsgBox "Рекомендуется снизить температуру!"
End If
```

Создание методов

Методы представляют собой обычные процедуры типов Sub и Function, которым выпало разместиться в модуле класса. Конечно, в большинстве случаев метод должен делать нечто, напрямую связанное с самим объектом, преобразуя данные, хранимые объектом. Но, при желании, в любой класс можно добавить и метод, рассчитывающий цены на прошлогодний снег.

VBA автоматически ассоциирует создаваемые вами методы с классами, в которые вы эти методы добавляете. Созданные вами методы можно вызывать из других частей программы точно так же, как и методы встроенных объектов.

Использование своих собственных объектов

Объекты, основанные на созданных вами классах, используются аналогично встроенным объектам VBA и объектам вашего приложения.

1. **Объявите переменную для объекта, например:**

   ```
   Dim objCustomThermostat As Thermostat
   ```

2. **Используйте оператор Set, чтобы создать реальный объект, с которым предполагается работать, например:**

   ```
   Set objCustomThermostat = New Thermostat
   ```

3. **Получите доступ к свойствам объекта или вызовите его методы, используя при этом стандартный синтаксис VBA, например:**

   ```
   objCustomThermostat.Setting = 65
   objCustomThermostat.CalculateEnergyUse
   ```

Использование элементов управления ActiveX

Вопреки сложившейся репутации программного империалиста, *Microsoft* стремится сделать свои средства разработки полностью "открытыми". Основываясь на спецификациях ActiveX, любой программист может создавать новые элементы управления, которые станут работать почти в любой программной среде, управляемой Windows, — включая C++, HTML, Visual Basic, **а** также VBA. Конечно, *Microsoft* определяет стандарты, которым должны соответствовать такие подключаемые программные единицы, так что от власти она ни в коей мере не отказывается.

В любом случае, в VBA можно добавить новые возможности, подключив дополнительные элементы управления ActiveX, не входящие в стандартный набор VBA. В одном проекте можно произвольно комбинировать элементы управления из разных источников. И все элементы управления ActiveX в основном работают так же, как и встроенные, которые, кстати, тоже являются элементами управления ActiveX.

В главе 21 приводится краткий обзор множества коммерческих и условно-бесплатных элементов управления ActiveX, которые помогут вам в работе. В настоящей главе обсуждаются общие приемы использования элементов управления ActiveX, а также даны некоторые советы об использовании элемента управления общим диалоговым окном (common dialog box control), предлагаемым Windows.

Некоторые элементы управления ActiveX правильно работают не в любой программной среде. Поэтому, прежде чем платить деньги за элемент управления, убедитесь, что он ведет себя в вашей версии VBA так, как обещает продавец. Вы должны также знать, что элементы управления ActiveX, созданные в Visual Basic, могут работать и ваших VBA-программах, но, вероятно, потребуют больших (около 2 Мбайт) файлов поддержки Visual Basic.

Добавление новых элементов в панель элементов управления

Чтобы получить возможность использовать элемент управления ActiveX, выполните следующее.

1. **Установите программное обеспечение элемента управления на жесткий диск.**
   Мне кажется, это имеет смысл.
2. **Зарегистрируйте элемент управления в Windows.**
   Зарегистрировать элемент управления можно несколькими способами, и, скорее всего, процедура установки сделает все за вас. Если же вам придется делать это самим, то работающему в VBA программисту проще всего использовать редактор Visual Basic.
3. **Выберите элемент управления, чтобы активизировать его для использования в VBA.**

Регистрация элемента управления

Чтобы зарегистрировать новый элемент управления, выясните имя файла, содержащего элемент управления, и место, где размещается этот файл на диске. После этого выполните следующее.

1. **Выберите Tools⇨References, чтобы открыть диалоговое окно со списком ссылок на элементы управления ActiveX, доступные для вашего проекта.**

2. **Щелкните на кнопке Browse (Обзор), чтобы открыть стандартное диалоговое окно Windows для открытия файлов.**
   Из списка в поле **Тип файлов** выберите ActiveX Controls (*.ocx).
3. **Найдите файл с новым элементом управления и двойным щелчком на файле откройте его.**
   Открытие файла вернет вас в диалоговое окно **References.**
4. **Найдите имя нового элемента управления в списке диалогового окна References и установите отметку соответствующего флажка.**
5. **Закройте диалоговое окно.**

Помещение элемента управления в панель Toolbox

Зарегистрировав новый элемент управления, активизируйте его, поместив элемент управления в панель элементов управления (Toolbox). Вот как это сделать.

1. **Активизируйте любое окно UserForm в редакторе Visual Basic, чтобы на экране появилась панель Toolbox.**
2. **Выберите Tools⇨Additional Controls из меню либо щелкните правой кнопкой мыши в панели Toolbox и выберите Additional Controls (Дополнительные элементы управления) из появившегося контекстного меню.**
3. **В списке доступных элементов управления найдите тот, который хотите активизировать, и установите соответствующий флажок.**
4. **Закройте диалоговое окно.**

Пиктограмма только что активизированного вами элемента управления должна появиться в панели Toolbox. На рис. 20.2 показана панель Toolbox, в которую добавили немало новых элементов управления.

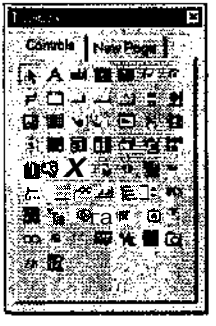


Рис. 17.2. Панель Toolbox с целым ворохом дополнительных элементов управления ActiveX

Если вы используете очень много дополнительных элементов управления, разместите их на дополнительных страницах в панели **Toolbox.** Чтобы добавить новые страницы в панель Toolbox, щелкните правой кнопкой мыши на ярлыке вверху вкладки и в появившемся меню выберите New Page (Новая страница), точно так же, как в случае элемента управления формой с множеством страниц.

Использование элементов управления ActiveX в программах

Добавив элемент управления ActiveX в панель Toolbox, вы можете добавлять его в свои формы точно так же, как стандартные элементы управления VBA. Правда, чтобы заставить элемент управления делать что-нибудь полезное, нужно знать, как работают его свойства и методы. Для этого вам понадобится документация и файлы справки, входящие в комплект поставки элемента управления. Если элемент управления спроектирован и установлен правильно, вы получите справку о любом из его свойств, нажав <F1>, когда нужное свойство выделено в окне свойств.

Глава 21

Три десятка ресурсов VBA

В этой главе...

- Дополнительная информация о VBA, представленная разработчиками программного обеспечения
- Периодические издания, посвященные VBA
- Поиск Web-ресурсов, относящихся к VBA
- Как получить новые элементы управления ActiveX и программные надстройки

Когда вы будете уверенно работать с VBA, у вас возникнет желание совершенствоваться. Эта глава укажет вам направления поиска новой информации и программных средств, которые помогут вам стать профессионалом. (Указанное в названии главы число может не совпадать с реальным числом ресурсов, упомянутых в этой главе, — на самом деле я никогда их не считал.)

Первая справочная инстанция

Программирование в VBA на самом деле несложно, по крайней мере, это относится к работе с переменными, управлению потоком выполнения программы и отображению форм. Трудности вероятны только при освоении объектной модели VBA-приложения. Необходимо знать, какие объекты нужны программе, как правильно на эти объекты сослаться и какие свойства и методы использовать, — иначе вас ждут разочаровывающие блуждания по справочной системе. Если нужна помощь по вопросам использования объектов VBA-приложения, первым делом обратитесь к разработчикам этого приложения.

Возьмите готовый программный код

Разработчики любого VBA-приложения всегда в том или ином виде предлагают помощь по поводу использования возможностей приложения и его объектной модели в пользовательских программах. Обязательно посмотрите, есть ли в файлах справки вашего приложения примеры программного кода. Часто на Web-страницах разработчиков можно найти тексты целых программ-примеров. Все это можно использовать как прекрасную стартовую площадку для создания своих процедур и программ.

Ознакомьтесь с предложениями Microsoft

Неужели после прочтения этой книги вы нуждаетесь в *дополнительной* информации по VBA? Ну что ж, тогда подумайте о покупке документации по Microsoft Visual Basic, которая представляет собой просто печатную копию файлов справки, но многие предпочитают иметь справочную информацию на бумаге, а не на экране.

На Web-странице сети *Microsoft* для разработчиков вы найдете много других ссылок и материалов о программировании в VBA. Вот адрес главной страницы, посвященной VBA:
`msdn.microsoft. com/vba/`

Не поленитесь также заглянуть на страницу, посвященную Visual Basic:
`msdn.microsoft.com/vbasic/`

Microsoft предлагает вашему вниманию и специальную страничку для разработчиков VBA-прооргам для приложений Office:
`msdn.microsoft.com/office/`

Разделы, посвященные конкретным приложениям Office и Microsoft Knowledge Base (База знаний Microsoft), предлагают статьи с описаниями приемов программирования и подводных камней, о которых следует знать программисту.

Журналы и газеты

До сих пор печатная страница предлагает самый удобный способ доступа к справочной информации. Журналов и газет по VBA великое множество. Типичный номер включает обзоры элементов управления ActiveX и других средств программирования, советы и приемы, помогающие программисту принимать правильные решения в сложных ситуациях, а также исходные тексты программ. По большей части эта информация применима и к VBA, а нередко публикуются статьи, в которых рассматриваются специальные вопросы программирования в VBA.

Обратите внимание на следующие печатные издания.

- ✓ *Microsoft Office & Visual Basic for Applications Developer*
  `www.officevba.com`
- ✓ *Visual Basic Programmer's Journal*
  `www.windx.com`
- ✓ *Inside Visual Basic*
  `www.elementkjournals.com/ivb/`

Web-страницы, посвященные VBA

Одним из способов убедить босса в том, что вы работаете не покладая рук, но при этом не затрачивать много усилий — это проводить часы подготовительной работы в Internet. Тысячи Web-страниц посвящены Visual Basic, и можно неделями постигать темные секреты программирования и собирать по сети элементы управления. Вы можете убедить босса в том, что когда вы наконец приступите к написанию программного кода, ваши исследования не пропадут даром и неимоверно повысят производительность вашего труда.

Web-страницы для программистов обычно просто изобилуют ссылками на другие относящиеся к рассматриваемой теме страницы. Начав со следующих адресов, у вас наверняка не будет проблем с тем, чем занять себя на долгое время:

`www.vbapro.com/`
`www.mvps.org/vbnet/`
`www.vbcity.com/`
`www.download.cnet.com` (здесь следует обратиться к ссылке Software)
`www.geocites.com/WallStreet/9245/`
`odyssey.apana.org.au/~abrowne/homepage.html` (для

разработок в Access)
www.slipstick.com/dev (для разработок в Outlook)
www.outlookexchange.com

Наверное, вы захотите ознакомиться и с Web-страницами тех журналов, газет и разработчиков программного обеспечения, о которых упоминается в данной главе.

Галактика элементов управления ActiveX

По последним подсчетам, число элементов управления ActiveX уже превысило число звезд во Вселенной. Относительно всего, что вы только можете пожелать вставить в свою VBA-программу, почти наверняка можно сказать, что кто-то уже разработал подходящий элемент управления ActiveX, способный выполнить всю работу за вас. У вас есть возможность выбирать из неимоверного количества коммерческих элементов управления ActiveX и еше более неимоверного количества условно-бесплатных в Web. И если вы действительно не можете найти подходящий элемент управления, то вполне вероятно, что вы нащупали коммерческую возможность, — подумайте о самостоятельном создании подходящего элемента управления и о предложении его массам.

Мания усовершенствования

Перед тем как потратить неопределенное количество дней на собирание элементов управления по отдельности, подумайте о покупке целого пакета элементов управления. Многие разработчики программного обеспечения предлагают наборы элементов управления ActiveX. И многие такие наборы начинаются с улучшенных версий стандартных элементов управления, с большим числом опций форматирования и лучшими возможностями управления данными. Часто эти элементы управления дополняют совершенно новые элементы управления типа манометров, реостатов и всевозможных кнопок-регуляторов. Среди других обнаруживаются элементы управления, предназначенные для работы с электронными таблицами, организации информации в виде древовидных структур наподобие иерархии папок в проводнике Windows, отправки факсов и печатания, работы с реестром Windows и невидимые элементы управления для отсчета времени.

В качестве отправной точки рассмотрите предложения следующих компаний.

***Infragistics** Corp.*

www.protoview.com

***ComponentOne** LLC.*

www.shersoft.com

Изобразительное искусство

Несмотря на то, что VBA специально не предназначен для импортирования изображений в различных форматах и последующего отображения этих изображений в формах, он довольно неплохо справляется с этим. Но если VBA-программе придется работать с форматом, который она не распознает, и особенно если вы пожелаете добавить изображениям живости с помощью спецэффектов, вам понадобится набор элементов управления ActiveX для работы с графикой. Вот адресок.

LeadTools

www.leadtools.com

Диаграммы и графики

Разглядеть смысл, скрывающийся за бесконечными рядами чисел, может только специалист, а вот понять (или по крайней мере заявить, что понимает) гистограмму или круговую диаграмму может каждый. В VBA нет встроенных средств для создания диаграмм, но такие средства несложно добавить с помощью элементов управления ActiveX.

ProEssentials

www.gigasoft.com

teeChartPro

www.stema.com

3D Charting Toolkit

www.nevron.com

Текстовые документы и электронные таблицы

Текстовые поля хороши только тогда, когда необходимо обработать только одну-две строки, введенные пользователем. Если же от программы на выходе ожидается получить готовый форматированный документ, то, очевидно, одними текстовыми полями не обойтись.

Когда дело касается представления чисел пользователю и обеспечения манипулирования этими числами, встроенные возможности VBA оказываются менее чем скромными. Нужно быть истинным мазохистом и располагать уймой свободного времени, чтобы пытаться строить что-то наподобие электронной таблицы с помощью стандартных элементов управления VBA.

Если в вашей системе есть Word и Excel, то очевидным решением выглядит использование их объектов с помощью VBA. Не имеет значения, в каком VBA-приложении выполняется ваша программа, — можно активизировать документы Word или Excel и извлечь из них нужные данные в соответствии со спецификациями COM.

Правда, загрузка Word или Excel потребует времени, а кроме того, нет возможности полностью контролировать их экранные представления. Поэтому элементы управления ActiveX предлагают более быструю и управляемую альтернативу, к тому же интегрируемую непосредственно в формы. Попробуйте следующие.

Элементы управления электронными таблицами

Spread

www.fpoint.com

FormulaOne

www.tidestone.com

Элементы управления текстовыми документами

TXTextControl

www.subsystems.com

TEDeveloper'sKit

www.subsystems.com

Бери деньги, и вперед!

К множеству странных и удивительных вещей, которые можно делать с помощью элементов управления ActiveX, относятся и подтверждение покупок по кредитным карточкам через модем и прямое подключение к Internet. Если вы разрабатываете пользовательскую систему для стола заказов или точки розничной продажи, просто бросьте в форму элемент управления activeCharge. Его методы и свойства позволяют отослать в банк информацию о кредитной карточке покупателя и получить из банка заключение о кредитоспособности этого покупателя. Ясно, что использование пользовательской формы для обработки информации о кредитных карточках имеет смысл, только если вы разрабатываете полномасштабную систему баз данных. Если вы работаете в VBA, а не в Visual Basic, то, вероятнее всего, вы используете при этом Access.

PCCharge DevKit

www.gcsoftinc.com

Разработка элементов управления

Microsoft не просто разрешает сторонним производителям создавать элементы управления ActiveX — она призывает к этому. В стандартную поставку Visual Basic 5 и 6 входят средства, позволяющие создавать пользовательские элементы управления. Только знайте, что при распространении программ, использующих элементы управления, созданные с помощью Visual Basic, вместе с программами придется распространять и библиотеки поддержки Visual Basic. Не забудьте в комплект поставки включить инструкции об установке этих библиотек.

Разное

Кроме элементов управления ActiveX, вечно спешащего VBA-программиста пытаются соблазнить массой другого разнообразного программного обеспечения, сладкими речами, обещающими ускорить и упростить цикл создания программ, — за отдельную плату, конечно.

Вычислительная мощь

К сильным сторонам VBA можно отнести объектно-ориентированный подход и визуальное проектирование форм. А вот когда дело касается скорости вычислений, VBA оказывается позади других языков программирования. Поэтому, если вы пишете программы для ведомства космических исследований или департамента переписи населения, то обратите внимание на компилятор PowerBasic. В соответствии с данными одной совершенно беспристрастной экспертной комиссии (разработчиков PowerBasic), один и тот же программный код в PowerBasic выполняется в 4-20 раз быстрее, чем в Visual Basic.

PowerBasic нужно рассматривать не как альтернативу VBA, а как дополнение. VBA попрежнему используется для запуска самой программы, взаимодействия с объектами приложения и отображения форм. Но представляющий основные вычисления программный код при этом компилируется с помощью PowerBasic в специальные библиотеки DLL, которые VBA использует при необходимости, Хотя в PowerBasic не используется присущий VBA объектно-ориентированный подход, ядро языка PowerBasic в сущности идентично VBA, так что переучиваться не придется.

PowerBASIC Inc.

www.powerbasic.com

Помогите, помогите!

Заставить VBA-программу отображать пользовательские окна справки совсем несложно, но вот создать сами файлы справки уже не так просто — если *не* пользоваться помощью специальных средств.

Создание файла справки официальным путем предполагает утомительный труд с множеством пробных отладочных запусков программного кода, написанного скучным, непостижимым и невизуальным языком программирования. Никто этим путем не идет. Лучше последовать за всеми и воспользоваться специальной программой, которая автоматически конвертирует созданный вами в обычном текстовом процессоре файл в готовый файл справки. При этом вы сможете конструировать свой файл справки визуально, во многом подобно тому, как создаются формы в VBA. К современным средствам, предлагающим создание как "классических" файлов справки Windows, так и файлов справки в формате HTML, относятся следующие.

RoboHelp

`www.blue-sky.com`

EasyHelp/Web и EasyHTML/Help

`www.eon-solutions.com`

Предметный указатель

И

К

М

Н

О

П

Р

С

T

У

Ф

Ц

Ч

Э

Я

Научно-популярное издание

Стив Каммингс

VBA для "чайников", 3-е издание

В издании использованы карикатуры американского художника Рича Теннанта

Литературный редактор *Л.Н. Важенина*
Верстка *В.И. Бордюк*
Художественный редактор *Е.П. Дынник*
Технический редактор *Г. И. Горобец*
Корректор *Л.А. Гордиенко, О.В. Мишутина*

Издательский дом "Вильямс".
101509, Москва, ул, Лесная, д. 43, стр. 1.
Изд. лиц. ЛР № 090230 от 23.06.99
Госкомитета РФ по печати.

Подписано в печать 12.12.2001. Формат 70×100/16.
Гарнитура Times. Печать офсетная.
Усл. печ. л. 36,12. Уч.-изд. л. 27,34.
Тираж 5000 экз. Заказ № 2347.

Налоговая льгота - общероссийский классификатор продукции ОК 005-93,
том 2: 953000 - книги и брошюры.

Отпечатано с диапозитивов в ФГУП "Печатный двор"
Министерства РФ по делам печати,
телерадиовещания и средств массовых коммуникаций.
197110, Санкт-Петербург, Чкаловский пр., 15.

Типы данных УВД

Тип данных	Содержимое соответствующей переменной	Диапазон допустимых значений
Boolean	Логические Истина или Ложь	Истина (-1) или Ложь (0)
Byte	Достаточно малое целое число	От 0 до 255
Integer	Не слишком большое целое число	От -32768 до 32767
Long	Большое целое число	От -2147483648 до 2147483647
Single	Значение одинарной точности с плавающей запятой	От -3,402823E38 до -1,401298E-45 для отрицательных значений и от 1,401298E-45 до 3,402823E38 — для положительных
Double	Значение двойной точности с плавающей запятой	От -1,79769313486232E308 до -4,94065645841247E-324 для отрицательных значений и от 4,94065645841247E-324 до 1,79769313486232E308 — для положительных
Currency	Большое число, для которого выделено 19 позиций, в том числе фиксированные четыре позиции после запятой	От -922337203685477,5808 до 922337203685477,5807
Decimal	Еще большее число, для которого выделено 29 позиций, в том числе до 28 позиций включительно для значения дробной части числа	Допустимый диапазон зависит от числа знаков после запятой, например: +/-79228162514264337593543950335 для чисел без дробной части или +/-7,9228162514264337593543950335 для чисел с 28 знаками после запятой
Data	Дата и время	От 1 января 100 г. до 31 декабря 9999 г.
Object	VBA-обьект	Ссылка на любой объект
String (переменной длины)	Последовательность переменной длины, состоящая из символов	Строковая переменная переменной длины мажет содержать от 0 примерно до 2 млрд. символов
String (фиксированной длины)	Последовательность заданной длины, состоящая из символов	Строковая переменная фиксированной длины может содержать от 0 примерно до 65400 символов
Variant	Любое из допустимых данных	Зависит от типа данных, содержащихся в переменной, в соответствии с приведенными выше описаниями
User-defined	Группы переменных, используемых вместе как единое целое	Зависит от типа переменных в группе в соответствии с приведенными выше описаниями

VBA для "чайников"™

COMPUTER
BOOK
SERIES

СЕРИЯ
КОМПЬЮТЕРНЫХ
КНИГ ОТ
ДИАЛЕКТИКИ

Шпаргалка

Работа с программным кодом

Для того чтобы...	Нажмите
Перейти к определению обьекта в точке ввода	<Shift+F2>
Открыть диалоговое окно поиска	<Ctrt+F>
Найти далее (найти, где следующий раз появляется текст, заданный в окне поиска)	<F3>
Найти предыдущее	<Shift+F3>
Заменить	<Ctrl+H>
Перейти к предыдущей редактировавшейся строке	<Ctrl+Shift+F2>
Отменить действие	<Ctrl+Z>
Открыть список свойств/методов	<Ctrl+J>
Открыть список констант	<Ctrl+Shift+J>
Получить краткую справку о переменной или объекте в точке ввода	<Ctrl+I>
Отобразить информацию о параметрах функции в точке ввода	<Ctrl+Shift+I>
Автоматически дополнить печатаемое слово	<Ctrl+пробел>
Перейти в о<не свойств к следующему свойству, начинающемуся с заданной буквы	<Ctrl+заданная буква>
Выполнить процедуру или программный код пользовательской формы, окно которой активно	<F5>
Приостановить выполнение программного кода и перейти в режим паузы	<Ctrl+Break>

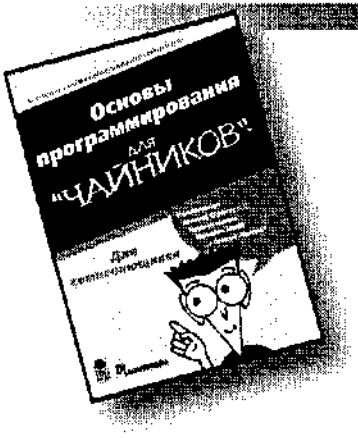
Отладка

Для того чтобы...	Нажмите
Начать выполнение программного кода последовательно по одной строке (пошаговое выполнение)	<FS>
Начать пошаговое выполнение программного кода без входа в вызываемые процедуры	<Shift+F8>
Выйти из вызванной процедуры и вернуться к основной программе	<Ctrl+Shift+F8>
Начать выполнение программного кода с остановкой на строке, содержащей текстовый курсор	<Ctrl+F8>
Указать (назначить) оператор, который должен выполняться следующим	<Ctrl+F9> (предварительно поместив текстовый курсор в соответствующий оператор)
Выполнить программный код процедуры обработки ошибки или вернуть ошибку в вызывающую процедуру	<Atl+F5>
Войти в процедуру обработки ошибки или вернуть ошибку в вызывающую процедуру	<Alt+F8>
Поместить точку останова в строку кода, содержащую текстовый курсор	<F9>
Убрать все точки останова	<Ctrl+Shift+F9>

Отображение окон

Для того чтобы...	Нажмите
Открыть окно программного кода для данной формы или элемента управления	<F7>
Открыть окно формы, соответствующей активному окну программного кода	<Shift+F7>
Перейти в следующее окно программного кода или формы	<Ctrl+Tab>
Открыть окно обозревателя обьектов	<F2>
Открыть окно немедленного выполнения команд	<Ctrl+G>

Hungry Minds™



Воллес Вонг

Основы программирования для "чайников"

В продаже

Стать настоящим программистом мечтает практически каждый пользователь компьютера. Однако определить, с чего следует начинать изучение программирования, а также какой язык программирования выбрать, очень непросто. В подобной ситуации не обойтись без рекомендаций настоящих профессионалов. Помочь начинающим программистам разобраться в таком многогранном мире программирования — такую задачу и поставил перед собой известнейший автор Воллес Вонг, приступая к работе над настоящей книгой. Простым человеческим языком автор расскажет вам, что такое программирование вообще, что такое язык программирования, как создаются компьютерные программы и какие инструменты для этого используются. Постепенно автор познакомит вас с основными премудростями написания компьютерных программ. Большое внимание в книге уделяется одному из самых популярных и простых для изучения языков программирования — BASIC. Кроме того, рассмотрено и использование таких языков программирования, как C/C++, C#, Pascal, Delphi, JBuilder, Java, JavaScript и некоторые другие. Вы узнаете, как организовать ввод и вывод данных, как использовать в программах переменные, константы и комментарии, познакомитесь с основными правилами использования чисел и строк, а также управляющими конструкциями и циклами. Кроме того, вы узнаете, что такое подпрограммы и каким образом они упрощают написание больших программ, как создавать графические изображения и заставить компьютер воспроизводить звуки, как сохранять данные в файлах, как создать интерфейс пользователя и отладить работу программы. Отдельные части книги посвящены использованию различных структур данных, а также программированию для Internet. Так, в настоящей книге рассматриваются такие темы, как использование массивов и записей для хранения данных, использование указателей, а знакомство с объектно-ориентированным программированием. Не забыты и такие важные вопросы, как сортировка и поиск данных, а также оптимизация кода программ. Помимо всего прочего, рассматриваются и использование HTML, создание интерактивных Web-страниц с помощью JavaScript и использование Java-аплетов. В конце книги вы найдете традиционные для серии "... для чайников" великолепные десятки.

Книга рассчитана на пользователей с начальным уровнем подготовки. Легкий и доступный стиль изложения поможет новичкам как можно быстрее приступить к созданию собственных программ.

Эмили Вандер Вер

JavaScript
для "чайников"

В продаже

Книга представляет собой пособие по основам программирования на языке сценариев JavaScript. В ней рассмотрены базовые средства JavaScript и методы их использования для решения конкретных задач, возникающих при разработке Web-страниц. Читатель узнает, как сделать Web-страницы более привлекательными, динамичными и как научить их "разговаривать" с посетителями ваших Web-узлов. Основные конструкции JavaScript вводятся с использованием примеров, которые иллюстрируют их применение на практике. В основу изложения материала каждой из глав положены один-два содержательных примера, снабженных подробными комментариями, показывающими действие операторов JavaScript. Примеры, приведенные в книге, представляют ценность сами по себе, поскольку охватывают наиболее распространенные задачи "интеллектуализации" Web-страниц и после небольших изменений их можно применять в качестве базиса для создания реальных Web-узлов.

Для тех, кто захочет познакомиться с JavaScript более глубоко, книга содержит описание объектной модели документа г учетом особенностей ее реализации в Web-броузерах Microsoft Internet Explorer и Netscape Navigator. Помимо прочего, книга содержит множество Web-адресов, по которым можно отыскать интересующую вас информацию по JavaScript.
Простота и ясность изложения, реальные, работающие примеры делают эту книгу незаменимой для тех, кто приступает к изучению JavaScript; книга также содержит информацию, которая во многом будет полезной и для более опытных пользователей HTML.

Эллен Финкельштейн, Гарди Лит

Flash 5 для "чайников"

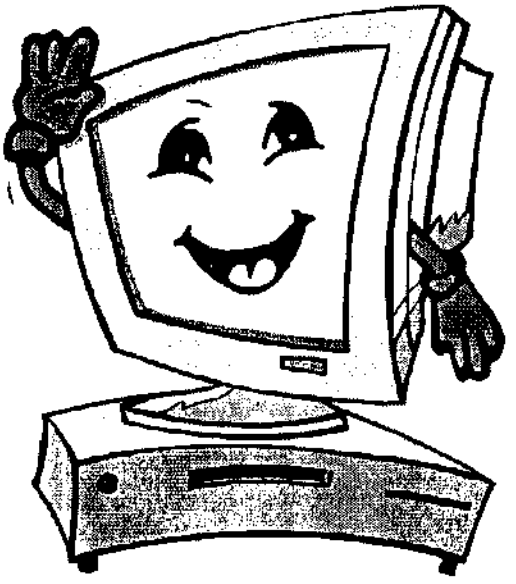
В продаже

Эта книга посвящена мощной программе для анимации, созданной компанией Macromedia, — Flash 5. Это последняя версия популярного программного обеспечения, которое используется многими Web-уэлами в Internet. Книга станет вам добрым другом и помощником в мире анимации. Здесь вы найдете объяснения человеческим языком, как с помощью Flash создать великолепные анимационные файлы для Web-узла. Здесь вы найдете всю необходимую информацию для того, чтобы немедленно приступить к работе с программой Flash.

В книге рассмотрен широкий спектр вопросов начиная с основных принципов работы с экраном программы Flash, панелями инструментов и командами меню, изучения возможностей программы до непосредственного создания анимации (основной задачи Flash) и кнопок для Web-страниц, добавление к фильмам звука, создание интерактивных Web-узлов и многое другое.

Эта книга проиллюстрирована множеством прекрасных примеров использования Flash, которые вы сможете найти в Internet. Она даст вам в руки инструменты, необходимые для самостоятельного создания анимированных Web-узлов.
Это идеальное руководство для начинающих пользователей, которые только знакомятся с программой Flash, или для тех, кому уже приходилось работать с ней и кто хочет отшлифовать свое мастерство.

домашний компьютер



Компьютер-этосовсемне сложно,аоченьинтересно. Мы подскажем, как подружиться со своим компьютером,каксделать егопонятнымиполезным для досуга, образования и работы.

журнал современнойсемьи

Подписной индекс: 3 4 2 8 8

Как подписаться на журнал «Домашний компьютер», см. на обороте

На журнал "Домашний компьютер" вы можете оформить
1. Онлайн подписку - **http://www.computerra.ru/site/subscription/**
2. Подписку по Объединенному каталогу в любом отделении связи России
3. Редакционную подписку

Чтобы подписаться через редакцию необходимо:
Оплатить в Сбербанке или любом другом банке, а также почтовым переводом сумму, соответствующую изданию и сроку подписки

Заполнить прилагаемый бланк заказа и копию платежного документа и выслать их:
либо по адресу **117419, Москва, 2-й Рощинский проезд, 8, ЗАО "Компьютерная пресса"**;
либо по факсу **(095) 956-19-38**;
либо отсканированную копию квитанции (в формате JPEG объемом до 100 Кб) по e-mail: **podpiska@computerra.ru**

Извещение

ЗАО «Компьютерная пресса.»
(получатель платежа)

ИНН получателя платежа 7729340216

расчетный счет 40702810100090000217

в АК "Московский муниципальный банк - Банк Москвы»

кор. счет 30101810500000000219

БИК 044525219

Вид платежа	Сумма
Заказываю предыдущих номеров журнала «Домашний компьютер»	
Подписка на «Домашний компьютер» на месяцев	
Всего	

Кассир

Дата Плательщик

Квитанция

ЗАО «Компьютерная пресса»

ИНН получателя платежа 7729340216

расчетный счет 40702810100090000217

в АК "Московский муниципальный банк - Банк Москвы"

кор. счет 30101810500000000219

БИК 044525219

Вид платежа	Сумма
Заказываю предыдущих номеров журнала «Домашний компьютер»	
Подписка на «Домашний компьютер» на месяцев	
Всего	

Кассир

Дата Плательщик

ДОСТАВОЧНАЯ КАРТОЧКА

НА ЖУРНАЛ

(почтовый индекс)	(код издания)
	34288

(код издания)

Домашний компьютер

на 200_ год по месяцам

1	2	3	4	5	6	7	8	9	10	11	12

ПВ | место | литер

Куда

(почтовый индекс) (адрес)

Кому

(фамилия, инициалы)