ГЛАВА 3

Методы разработки приложений

Эта глава располагается в начале книги, поскольку в ней рассматривается материал, без знания которого вам будет сложно изучать следующие главы. К сожалению, в этой главе содержится несколько разделов, темы которых так и не будут раскрыты. Таким образом, для получения более полной картины вам придется заново перечитать эту главу после того, как изучите всю книгу.

Знакомясь с материалом данной главы, следует помнить, что хотя описанные в ней методы и считаются общепринятыми, они не всегда достойны практического применения в собственных приложениях. Из этой главы вы узнаете о самых распространенных приемах и методах, описанных в следующих главах.

Соглашения об именовании

Важность соглашения об именовании

Соглашение об именовании используется при присвоении имен различным элементам приложения. Всякий раз при объявлении переменной или создании пользовательской формы мы определяем имя. Объекты неявно именуются в результате принятия имени, предложенного по умолчанию, даже если вы об этом и не подозреваете, например, при создании пользовательской формы. Одним из правил хорошего программирования является безоговорочное следование принципам объявленного ранее соглашения об именовании элементов во всем VBA-приложении.

Взгляните на пример, который демонстрирует суть соглашения об именовании. Что мы знаем о переменной x в следующем коде?

```
x = wksDataSheet.Range("A1").Value
```

Проанализировав место использования, мы можем предположить, что это переменная. Но какой тип данных она содержит? Какова ее область действия: общедоступная, уровня модуля или локальная? Каково ее назначение в программе? Мы не сможем ответить на эти вопросы, не потратив время на изучение остального кода. Придерживаясь правильного соглашения об именовании, вы получите ответы на эти вопросы, проанализировав имя переменной. Ниже приведен исправленный пример. (Подробно специфику именования элементов мы рассмотрим в следующем разделе.)

```
glListCount = wksDataSheet.Range("A1").Value
```

Теперь мы знаем область действия переменной (символ `g` обозначает глобальную или общедоступную переменную), ее тип данных (символ `l` соответствует типу данных `Long`) и примерно представляем ее назначение (хранит количество элементов в списке).

Соглашение об именовании помогает непосредственно распознавать тип и назначение отдельных частей приложения. Это позволяет разработчику сконцентрироваться на изучении кода, а не на анализе его структуры. Соглашение об именовании способствует самодокументированию кода, а также уменьшает количество комментариев, вводимых для описания операций, выполняемых в коде.

В следующем разделе приведен пример хорошо продуманного соглашения об именовании. Самое важное в соглашении об именовании заключается в том, что вы принимаете его один раз и используете постоянно. До тех пор, пока каждый участник проекта понимает соглашение об именовании, не имеет особого значения, какие префиксы вы используете или в каком регистре вводятся имена элементов (строчные, прописные и т.д.). При принятии соглашения об именовании его логические правила используются в дальнейшем на протяжении всей разработки проекта.

Пример соглашения об именовании

Правильное соглашение об именовании применяется по отношению не только к переменным, но и ко всем остальным элементам приложения. Пример соглашения об именовании, представленный ниже, применяется при именовании всех элементов типичного приложения Excel. Мы начнем с рассмотрения переменных, констант и связанных с ними элементов, поскольку это общие объекты для любого приложения. В табл. 3.1 показан основной формат соглашения об именовании. Отдельные элементы соглашения об именовании и их назначение будут описаны ниже.

Таблица 3.1. Соглашение об именовании для переменных, констант, определенных пользователем типов и перечисляемых элементов

Элемент	Соглашение об именовании
Переменные	*<область_действия><массив><тип_данных>Описательное_Имя*
Константы	*<область_действия><тип_данных>ОПИСАТЕЛЬНОЕ_ИМЯ*
Определяемые пользователем типы	`Type` *ОПИСАТЕЛЬНОЕ_ИМЯ* *<тип_данных>Описательное_Имя* `End Type`
Перечисляемые элементы	`Enum` *<префикс_проекта>Общее_описание* *<префикс_проекта>ОбщеОписательное_имя1* *<префикс_проекта>ОбщеОписательное_имя2* `End Enum`

Спецификатор области действия

- `g` — общедоступная (глобальная)
- `m` — уровня модуля
- (Ничего) — уровня процедуры

Спецификатор массива

- `a` — массив
- (Ничего) — не массив

Спецификатор типа данных

Существует настолько много типов данных, что для них довольно сложно составить полный список префиксов. Со встроенными типами данных все просто. В большинстве часто используемых встроенных типов данных применяются краткие префиксы. Проблемы возникают при именовании переменных объектов, которые относятся к различным приложениям. Некоторые программисты используют префикс `obj` в именах всех объектов. Это недопустимо. При этом разработка последовательных, уникальных и коротких префиксов для каждого типа объектов также вызывает много проблем. Пытайтесь использовать максимально понятные префиксы из двух-трех букв для наиболее часто используемых переменных объектов, а префикс `obj` оставьте для объектов, которые появляются в вашем коде нечасто.

Постарайтесь сделать программный код максимально простым и, в первую очередь, согласованным. Используйте префиксы типов данных, состоящие из трех (или меньшего количества) символов. Длинные префиксы в комбинации со спецификаторами области действия и массива приводят к получению длинных и малопривлекательных имен переменных. В табл. 3.2 показаны примеры префиксов для наиболее распространенных типов данных.

Использование описательных имен

В VBA имя переменной может состоять не более чем из 255 символов. Не стоит давать элементам длинные имена, но также не старайтесь присваивать переменным слишком короткие имена, поскольку со временем код станет слишком сложным для понимания как вами, так и всеми, кто будет с ним работать.

Интегрированная среда разработки Visual Basic предоставляет возможность автоматически завершать идентификаторы (все имена, используемые в приложении). Вам нужно набрать всего лишь несколько первых символов, чтобы ввести требуемое имя. Введите несколько первых символов имени и нажмите комбинацию <Ctrl+пробел> для открытия списка автоматического завершения имен, которые начинаются с указанных символов. При вводе дополнительных символов список будет уменьшаться. На рис. 3.1 комбинация клавиш <Ctrl+пробел> использована для отображения списка строчных констант сообщений, которые можно добавлять в окно сообщения.

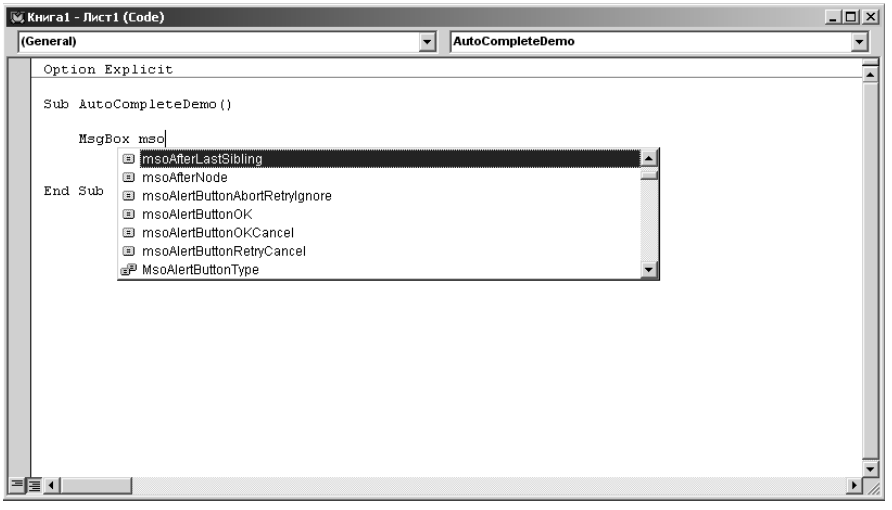

Рис. 3.1. Использование комбинации <Ctrl+пробел> для автоматического завершения длинных имен

Таблица 3.2. Предлагаемые префиксы соглашения об именовании

Префикс	Тип данных	Префикс	Тип данных	Префикс	Тип данных
b	Булев	cm	ADODB.Command	cbo	MSForms.ComboBox*
byt	Байтовый	cn	ADODB.Connection	chk	MSForms.CheckBox
cur	Денежный	rs	ADODB.Recordset	cmd	MSForms.CommandButton
dte	Дата			ddn	MSForms.ComboBox**
dec	Десятичный	cht	Excel.Chart	fra	MSForms.Frame
d	Двойной точности	rng	Excel.Range	lbl	MSForms.Label
i	Целочисленный	wkb	Excel.Workbook	lst	MSForms.ListBox
obj	Объект		opt	MSForms.OptionButton	
sng	Единичной точности	cbr	Office.CommandBar	spn	MSForms.SpinButton
s	Строка	ctl	Office.CommandBarControl	txt	MSForms.TextBox
u	Определяемый пользователем				
v	Переменный	cls	Переменная класса, определенного пользователем	ref	RefEditControl
		frm	Переменная пользовательской формы	col	VBA.Collection

* Используется в элементах управления ComboBox, имеющих тип DropDownComboStyle.

** Используется в элементах управления ComboBox, имеющих тип DropDownListStyle.

Замечания о перечисляемых типах данных

Перечисляемый тип данных представляет собой специальный тип констант, доступных в Excel 2000 и более поздних версиях. Он позволяет объединять похожие по своей структуре имена в список связанных значений. В объектных моделях VBA и Excel перечисляемые типы данных применяются очень часто. Вы можете ознакомиться с ними, воспользовавшись функцией автоматического завершения имен, которая применяется в VBA по отношению к значениям многих свойств. Например, если вы введете в модуле VBA код

```
Sheet1.PageSetup.PaperSize =
```

то на экран будет выведен длинный список перечисляемых констант `XlPaperSize`, представляющий форматы бумаги, доступные для печати. На рис. 3.2 показан этот список в полном размере.

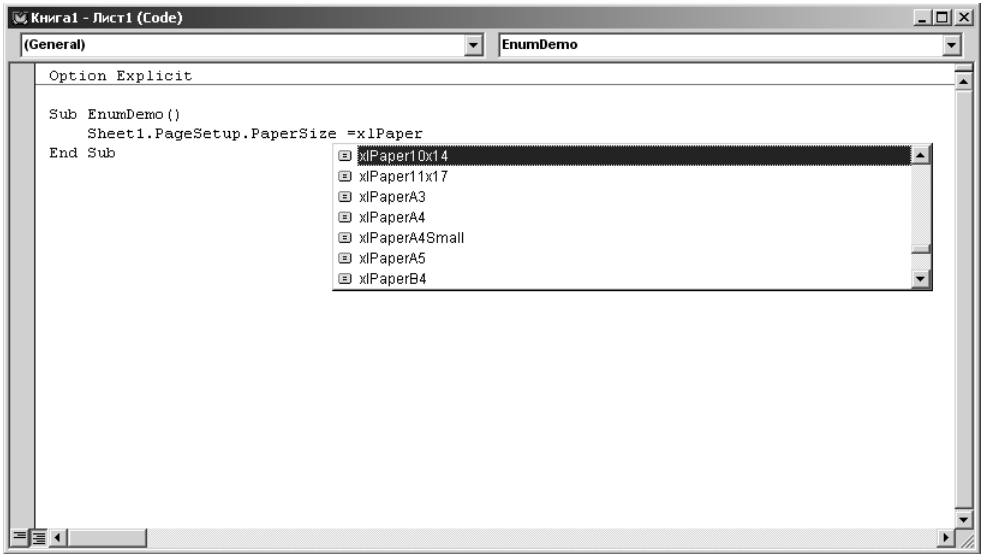

Рис. 3.2. Список перечисляемых форматов бумаги, поддерживаемых в Excel

Эти имена реально представляют числовые константы, со значениями которых вы можете ознакомиться в окне Object Browser, описанном в главе 16. Изучите структуру этих перечисляемых значений. Во-первых, все они начинаются с префикса, идентифицирующего приложение, к которому они относятся. В данном случае используется префикс `xl`, т.е. имена представляют элементы программы Excel. Во-вторых, первая часть имени представлена термином, описывающим перечисляемые значения, в данном случае это `Paper` (Бумага). Остальная часть каждого имени перечисляемого типа данных представляет уникальную строку, описывающую каждое конкретное значение. Например, имя `xlPaper11x17` представляет формат бумаги 11×17, а имя `xlPaperA4` — формат бумаги A4. Эта система именования перечисляемых констант довольно популярна и, в частности, применяется в настоящей книге.

Примеры соглашений об именовании

Соглашения об именовании — это весьма размытое понятие, которое сразу сложно связать с реальными именами. В этом разделе мы попытаемся сопоставить некоторые соглашения об именовании с реальными именами. Все примеры взяты непосредственно из коммерческих приложений, написанных авторами книги.

Переменные

- **gsErrMsg**. Общедоступная переменная с типом данных `String`, используемая для хранения сообщений об ошибках.
- **mauSettings()**. Массив уровня модуля, определяемого пользователем, содержащий список настроек.
- **cbrMenu**. Локальная переменная с типом данных `CommandBar`, содержащая ссылку на строку меню.

Константы

- **gbDEBUG_MODE**. Глобальная константа типа `Boolean`, указывающая, отлаживается ли проект в настоящее время.
- **msCAPTION_FILE_OPEN**. Константа уровня модуля с типом данных `String`, содержащая заголовок диалогового окна открытия файла, настраиваемого пользователем (в данном примере — `Application.GetOpenFilename`).
- **lOFFSET_START**. Локальная константа с типом данных `Long`, содержащая точку, из которой начинает отсчитываться смещение относительно объекта `Range`.

Типы данных, определяемые пользователем

Ниже приведен определенный пользователем тип данных, который используется для хранения размера и положения объекта. Он состоит из четырех переменных с типом данных `Double`, в которых указываются ширина, высота и координаты верхнего левого угла объекта, а также из переменной типа `Boolean`, определяющей необходимость сохранения этих настроек.

```
Public Type DIMENSION_SETTINGS
  bSettingsSaved As Boolean
  dValTop As Double
  dValLeft As Double
  dValHeight As Double
  dValWidth As Double
End Type
```

Переменные внутри кода описания пользовательского типа данных называются *переменными типа данных*. Их можно объявлять в любом порядке. Однако в нашем соглашении об именовании рекомендуется сортировать их в алфавитном порядке по стандартным типам данных, если нет особых причин группировать их иным способом.

Перечисляемые типы данных

Ниже приведен пример перечисляемого типа данных, используемый для описания различных дней в году. Префикс `sch` в имени перечисляемого типа использован для указания имени приложения. Данный перечисляемый тип используется в приложении с именем `Scheduler`. Строка `DayType` в имени перечисляемого типа данных указывает

на назначение этого типа, а каждый отдельный член перечисляемого типа данных имеет уникальный индекс, описывающий назначение члена.

```
Private Enum schDayType
   schDayTypeUnscheduled
   schDayTypeProduction
   schDayTypeDownTime
   schDayTypeHoliday
End Enum
```

Если вы не укажете, какие значения хотите присвоить членам перечисляемого типа, то VBA автоматически присвоит первому члену списка значение “нуль” и будет увеличивать его на единицу в каждом последующем члене. Вы можете изменить такое поведение редактора и назначить другую точку отсчета, с которой VBA будет начинать приращение значений. Например, чтобы начать увеличение значений с единицы, а не с нуля, примените следующий код.

```
Private Enum schDayType
   schDayTypeUnsheduled = 1
   schDayTypeProduction
   schDayTypeDownTime
   schDayTypeHoliday
End Enum
```

VBA будет продолжать увеличивать значение в каждом следующем члене, которому оно присваивается. Вы можете отменить автоматическое присвоение значений всем членам перечисляемого типа данных, присвоив им другие значения вручную.

На рис. 3.3 показано главное преимущество перечисляемых типов данных. VBA позволяет автоматически заполнять список значений для любой переменной, объявленной с перечисляемым типом данных.

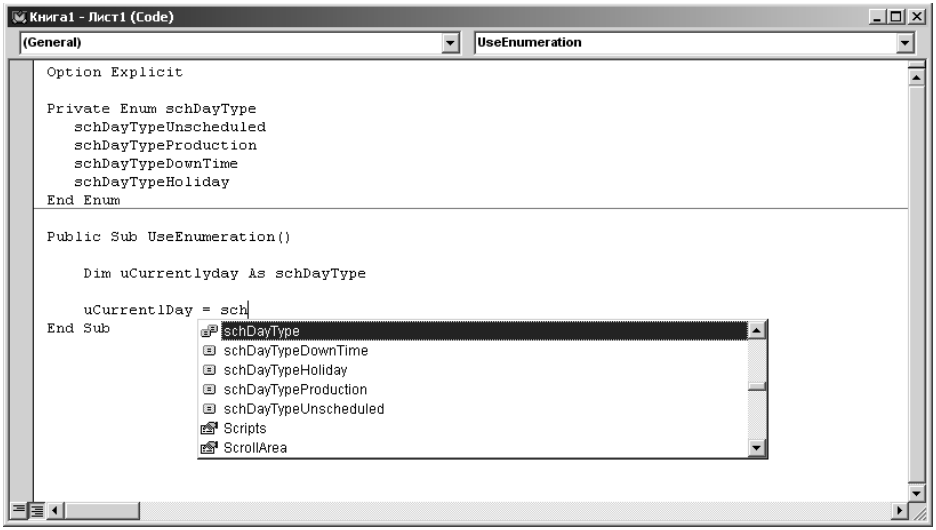

Рис. 3.3. Даже при использовании собственных перечисляемых типов данных вы имеете возможность автоматически заполнять список значений в VBA

Процедуры

Подпрограммы и функции VBA объединяются под общим названием *процедуры*. Всегда присваивайте процедурам описательные имена. Опять-таки, вы можете использовать в имени процедур до 255 символов, а имена процедур добавляются в список автоматического завершения имен элементов, открываемый с помощью комбинации клавиш <Ctrl+пробел>. Поэтому нет особого смысла укорачивать имена процедур, делая их малопонятными и лишенными описательной информации.

Это, конечно, не самая распространенная практика, однако мы считаем, что включение в имя функции префикса, указывающего на тип данных возвращаемого ими значения, упрощает понимание программного кода. При именовании функции всегда используйте открывающую и закрывающую круглые скобки в качестве отличительного признака от имени переменной или подпрограммы, даже если в функции аргументы не используются. В листинге 3.1 показана правильно именованная функция типа `Boolean`, используемая для проверки утверждения, заданного конструкцией `If...Then`.

Листинг 3.1. Пример соглашения об именовании функций

```
If bValidatePath("C:\Files") Then
   ' Блок If...Then выполняется,
   ' если существует заданный путь
End If
```

Процедурам следует давать имена, которые однозначно трактуют выполняемые ими задачи. Например, назначение процедуры с именем `ShutDownApplication` не вызывает особых сомнений. Функциям следует присваивать имена, описывающие возвращаемое ими значение. От функции с именем `sGetUnusedFilename()` вполне стоит ожидать возвращения имени файла.

Соглашение об именовании, применяемое по отношению к аргументам процедур, ничем не отличается от соглашения об именовании для переменных уровня модуля. Например, функцию `bValidatePath`, показанную в листинге 3.1, можно объявить следующим образом.

```
Function bValidatePath(ByVal sPath As String) As Boolean
```

Модули, классы и пользовательские формы

В нашем примере соглашения об именовании перед именами стандартных модулей кода вводится прописная буква `M`, перед именами классов — прописная буква `C`, а перед пользовательскими формами — прописная буква `F`. Таким образом упрощается операция сортировки этих объектов на панели Project (Проект) окна VBE. На рис. 3.4 эта панель показана в представлении папок.

Если придерживаться такого соглашения об именовании, то код, в котором используются классы и объекты пользовательских форм, становится максимально понятным. Например, в следующем коде такое соглашение об именовании делает максимально понятным такие операции, как объявление переменной объекта с определенным типом класса и создание экземпляра этого класса.

```
Dim clsMyClass As CMyClass
Set clsMyClass = New CMyClass
```

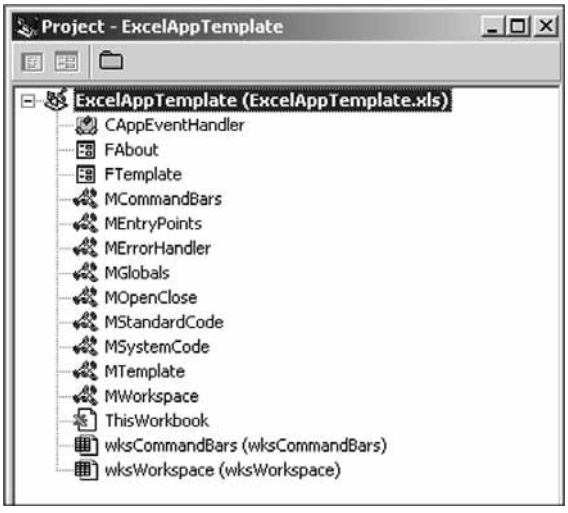

Рис. 3.4. Модули классов, пользовательские формы и стандартные модули на панели Project

В каждом случае имя слева представляет ***переменную*** класса, а объект справа — сам ***класс***.

Рабочие листы и диаграммы

Поскольку рабочие листы и листы диаграмм в вашем проекте интерпретируются VBA как переменные встроенных объектов, представляющих листы рабочей книги, их имена должны создаваться соответственно соглашению об именовании переменных. В именах рабочих листов в коде используется префикс `wks`, чтобы идентифицировать их как ссылки на объекты `Worksheet`. Подобным образом в именах листов диаграмм в коде используется префикс `cht` для определения их как ссылок на объекты `Chart` в Excel.

В именах обоих типов листов префикс вводится перед описательной частью имени, указывающей назначение листа в приложении. На рис. 3.4, например, показан рабочий лист `wksCommandBars`, содержащий таблицу объявлений панелей инструментов, созданных приложением. В именах листов надстроек и скрытых листов рабочей книги, не предназначенных для просмотра конечным пользователем, имя ярлыка листа вполне может совпадать с его именем в программном коде. Что касается листов, с которыми работают конечные пользователи, имя ярлыка листа должно быть максимально описательным, а также предназначенным для изменения пользователем. Как будет описано далее, при разработке приложения вы всегда должны полагаться исключительно на имена в коде, а не на имена ярлыков рабочих листов в VBA-коде.

Проект Visual Basic

Обратите внимание на то, что на рис. 3.4 проекту Visual Basic присвоено то же имя, что и рабочей книге, связанной с ним. Вы всегда должны присваивать проекту имя, четко идентифицирующее приложение, к которому относится этот проект. Если вы планируете связывать проекты ссылками, то им придется присваивать уникальные имена.

Соглашения об именовании элементов пользовательского интерфейса Excel

Элементы пользовательского интерфейса Excel, используемые в создаваемом приложении, также подпадают под правила соглашения об именовании. В предыдущем разделе мы рассмотрели правила именования рабочих листов и листов диаграмм. К остальным трем основным категориям элементов пользовательского интерфейса Excel относятся формы, встроенные объекты и именованные объекты.

Формы

Термин *форма* описывает группу самых разных объектов, которые размещаются в верхней части рабочего листа или листа диаграммы. Формы можно условно разделить на три категории: элементы управления, графические объекты и встроенные объекты. Именовать формы следует примерно так же, как и переменные объектов, т.е. перед именем должен вводиться префикс, указывающий тип объекта, а описательная часть имени должна представлять назначение формы в приложении.

Многие элементы управления, которые размещают в пользовательских формах, можно вставлять непосредственно на рабочие листы. Рабочие листы могут также содержать элементы управления, добавленные с помощью панели инструментов Формы и напоминающие объекты коллекции `MSForms` (элементы управления ActiveX), хотя и имеющие собственные преимущества и недостатки. В главе 4 эти вопросы описаны более подробно. Элементы управления, размещаемые на рабочих листах, именуются соответственно правилам соглашения об именовании, которые характерны для элементов управления, размещенных на пользовательских формах.

Рабочие листы также могут содержать широкий набор нарисованных объектов (называемых *фигурами*), которые, строго говоря, не являются элементами управления несмотря на то, что вы можете назначить им всем макросы. По отношению к ним применяются такие же правила соглашения об именовании, как и для большинства других объектов, используемых в VBA. Разработка уникальных префиксов для всех них может оказаться очень сложным занятием, так что вводите их только для самых распространенных графических объектов. Ниже приведены примеры префиксов для трех наиболее распространенных типов графических объектов.

- `pic` — рисунок
- `rec` — прямоугольник
- `txt` — текстовый блок или элемент управления ActiveX

Встроенные объекты

Под *встроенными объектами* понимают такие объекты Excel, как сводные таблицы и объекты диаграмм, а также объекты, созданные другими приложениями (отличными от Excel). Рабочие листы могут содержать самые разные встроенные объекты. Примерами встроенных объектов, отличных от объектов Excel, могут выступать формулы, созданные с помощью редактора Equation Editor, и художественные объекты WordArt. Ниже представлены префиксы для некоторых типов встроенных объектов.

- `cht` — объект диаграммы
- `eqn` — формула
- `qry` — таблица запроса

- pvt — сводная таблица
- art — объект WordArt

Именованные объекты

Соглашение об именовании объектов несколько отличается от подобного соглашения для других элементов программы. В случае с именованными объектами префикс указывает назначение объекта, а не тип данных, которые хранятся в текущем элементе. Причина данного требования заключается в том, что нетривиальные приложения Excel обычно управляют многими именованными объектами, которыми проще манипулировать, если они сгруппированы по целевым категориям в диалоговом окне Присвоить имя. Если рабочий лист содержит десятки, а то и сотни именованных объектов, то их сортировка по функциональной принадлежности, представляемой префиксом, только увеличит эффективность управления.

Описательная часть имени по-прежнему указывает назначение именованного объекта в определенной категории. В следующем списке приведены префиксы для наиболее распространенных типов именованных объектов.

- cht — диапазон данных диаграммы
- con — именованная константа
- err — флаг ошибки
- for — именованная формула
- inp — диапазон ввода
- out — диапазон вывода
- ptr — расположение определенной ячейки
- rgn — регион
- set — настройка пользовательского интерфейса
- tbl — таблица

Отказ от соглашения об именовании

Вы можете нарушить правила соглашения об именовании объектов в двух ситуациях. Первая ситуация — при управлении объектами, участвующими в вызове API-функций Windows. Имена этих объектов назначаются разработчиками компании Microsoft, а потому они хорошо известны многим программистам. Константы API-функций, пользовательские типы данных, объявления процедур и аргументы процедур должны представляться в коде приложения в том же виде, в котором они заданы в наборе инструментальных средств разработки программного обеспечения Microsoft (Microsoft Platform SDK), с которым можно ознакомиться на Web-узле базы знаний Microsoft.

```
http://msdn.microsoft.com/library/en-us/winprog/windows_api_start_page.asp
```

Обратите внимание, что данный ресурс содержит описание API-функций, созданных с использованием C/C++.

Вторая ситуация, в которой необходимо нарушить правила соглашения об именовании, — это применение “внешнего” кода из стороннего источника для выполнения специфических задач. При изменении имен, используемых в этом коде, и последующем обращении к соответствующим объектам в коде основного приложения вам будет сложно обновить “внешний” код (конечно, только в случае разработки его новой версии).

Организация и структурирование приложений

Структура приложения

Приложение с одной рабочей книгой и приложение с N рабочими книгами

Количество рабочих книг, используемых в приложении Excel, зависит от двух факторов: сложности самого приложения и уровня сложности распространения приложения. Простые приложения и приложения, которые устанавливаются нестандартным образом, требуют применения минимального количества рабочих книг. Сложные приложения и приложения, установка которых находится в полной компетенции конечных пользователей, могут содержать большое количество рабочих книг и файлов других типов, таких как DLL-библиотеки. В главе 2 описаны основные типы приложений Excel и их структура.

Если у вас имеется возможность разбить приложение на множество файлов, то эту задачу все же нужно выполнить по целому ряду причин. В частности, среди возможных причин можно выделить разделение приложения на несколько логических уровней, отделение кода от данных, отделение элементов пользовательского интерфейса от элементов кода, выделение функциональных средств приложения и управление конфликтами, вызываемыми изменениями, вносимыми командой разработчиков в процессе отладки.

Разделение логических уровней

Практически каждое нетривиальное приложение Excel разделяется на три отдельных логических уровня.

- **Уровень пользовательского интерфейса**. На уровне пользовательского интерфейса приложение состоит из кода и отображаемых на экране элементов, которые требуются для его взаимодействия с пользователем. В приложении Excel пользовательский интерфейс содержит такие элементы, как рабочие листы, листы диаграмм, панели инструментов, пользовательские формы и коды, требуемые для непосредственного управления этими элементами. Уровень пользовательского интерфейса является единственным логическим уровнем, содержащим элементы, представляемые пользователям.
- **Уровень программной логики, или уровень приложения**. Уровень программной логики представлен одним только кодом. Данный код выполняет основные операции приложения. На уровне программной логики принимаются входные данные, поступающие с уровня пользовательского интерфейса, и затем возвращаются выходные данные, опять-таки, на уровень пользовательского интерфейса. В случае выполнения длительных операций с уровня деловой логики на уровень пользовательского интерфейса могут передаваться отдельные обновленные данные (например, в форме сообщений в строке состояния или диаграммы выполнения операции).
- **Уровень хранения данных и доступа к ним**. Уровень хранения данных и доступа к ним отвечает, как следует из названия, за хранение и извлечение данных, требуемых для нормального функционирования приложения. Он может быть довольно простым, представляясь ячейками локального скрытого рабочего листа, в которых сохраняются значения, или же более сложным — в виде базы данных SQL Server,

доступ к которым выполняется через сетевое соединение с помощью специальных процедур. Уровень хранения данных и доступа к ним взаимодействует непосредственно только с уровнем деловой логики.

На рис. 3.5 показаны все три уровня, которые характеризуют завершенное приложение, хотя они не должны связываться неразрывно. Три уровня приложения связываются свободно, поэтому локальное изменение на одном уровне не требует редактирования данных на двух других уровнях. Жесткое связывание уровней неизбежно приводит к сложностям, вызванным последующей поддержкой и обновлением приложения.

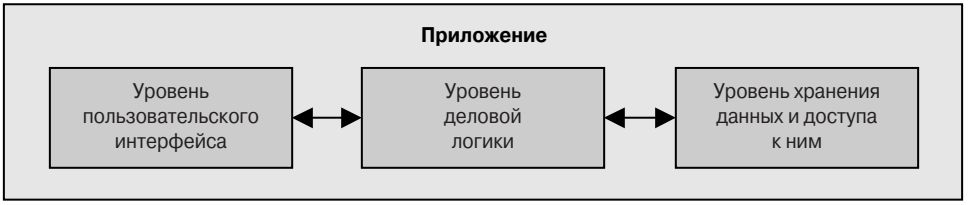

Рис. 3.5. Взаимосвязи между тремя уровнями приложения Excel

Например, если уровень хранения данных и доступа к ним требует перехода от базы данных Access к базе данных SQL Server, вы можете выполнить все необходимые изменения на уровне хранения данных и доступа к ним. В правильно спланированном приложении такие изменения никак не влияют на два других уровня. В идеальном случае обмен данными между уровнем программной логики и уровнем хранения данных и доступа к ним реализуется с помощью пользовательских типов данных. Это обеспечивает оптимальное сочетание производительности приложения и свободы связывания данных отдельных уровней. В качестве альтернативного решения можно предложить использовать объекты ADO `Recordset`, однако будьте готовы к решению проблем со связыванием данных разных уровней. Так что лучше, если уровень программной логики не полагается на такие характеристики, как порядок полей, возвращаемых из базы данных.

Подобным образом, если вам нужно создать презентационный интерфейс для Web-приложения, лучше всего использовать свободную связь между уровнем пользовательского интерфейса и уровнем программной логики. И все потому, что на уровне программной логики не делается никаких допущений о структуре и содержимом пользовательского интерфейса. Элементы, которые принимают данные, введенные пользователем, полностью автономны. С уровня программной логики на уровень пользовательского интерфейса информация, требуемая для инициализации, передается в виде свойств, содержащих данные простых типов. С уровня пользовательского интерфейса, запрашивающего входные данные у пользователя, информация передается обратно на уровень программной логики в виде значений свойств простых типов или в более сложных интерфейсах в формате UDT (User Defined Types —пользовательские типы данных). Поскольку на уровне программной логики ничего не известно о структуре пользовательского интерфейса, ссылки из процедуры уровня программной логики непосредственно на пользовательскую форму категорически запрещены.

Отделение данных и пользовательского интерфейса от кода

Уровень пользовательского интерфейса многих приложений Excel содержит два уникальных подуровня. Они состоят из рабочей книги и элементов листов, составляю-

щих основу пользовательского интерфейса, а также из кода поддержки этих элементов. Концепция отделения кода от данных распространяется в том числе и на эти подуровни. Интерфейс, разработанный на основе рабочей книги, не должен содержать программный код, а код графического интерфейса, контролирующий работу элементов управления рабочей книги, должен храниться в надстройке, полностью отделенной от рабочей книги, которым эта надстройка управляет.

Причина такого отделения такая же, как и в описанном выше случае разделения приложения на отдельные уровни. Заключается она в изолировании операций изменения данных. Из всех уровней приложения наиболее частым изменениям подвергается уровень пользовательского интерфейса. Поэтому недостаточно просто изолировать изменения в пользовательском интерфейсе на его уровне. Вы также должны изолировать изменения в отображаемых элементах пользовательского интерфейса от кода, управляющего интерфейсом.

Реальные примеры разделения уровней приложения приведены в последующих главах. Поэтому не отчаивайтесь, если на этом этапе вам не все понятно.

Подготовка приложения для процедурного программирования

Процедурное программирование представляет собой способ программирования, с которым знакомо большинство разработчиков. Оно включает в себя разбиение приложения на множество процедур, каждая из которых предназначена для выполнения в приложении специфических задач. С помощью процедур можно запрограммировать все приложение; элементы процедурного программирования можно эффективно сочетать с элементами объектно-ориентированного программирования. Отдельное приложение также можно создать с помощью одного только объектно-ориентированного программирования. В этом разделе описаны наилучшие методы процедурного программирования. Методы объектно-ориентированного программирования рассматриваются в главе 7.

Разделение кода на модули по функциям и категориям

Основная цель разделения кода на модули заключается в повышении надежности и удобства использования приложения. В процедурном приложении программный код логически организован в отдельные модули. Наилучший способ выполнения этой задачи состоит в группировании процедур, выполняющих подобные функции, в отдельные модули.

Примечание. VBA имеет незадокументированное ограничение на размер стандартного модуля с программным кодом. Стандартный модуль не должен занимать на диске больше 64 Кбайт (имеется в виду размер текстового файла с кодом при его экспорте из проекта). (Утилита `VBETools` на компакт-диске, прилагаемом к книге, автоматически отслеживает размеры модулей с кодом.) Ваш проект будет сохранять работоспособность при превышении этого ограничения (64 Кбайт), однако почти всегда это будет вызывать сбои в приложении.

Функциональное разделение

Операция функционального разделения приложения применяется по отношению к процедурам, позволяя каждой процедуре выполнять свою задачу. Теоретически вам никто не запрещает написать приложение, состоящее из одной гигантской процедуры.

Однако при этом ваше приложение будет сложно отлаживать и поддерживать. Выполнив функциональное разделение, вы структурируете свое приложение таким образом, что оно будет состоять из множества процедур, каждая из которых отвечает за правильность выполнения только определенной задачи, что облегчает понимание, проверку, документирование и поддержку проекта.

Методы создания процедур

Полный список инструкций по созданию процедур может занять целую главу. Ниже представлены самые важные из них.

- **Автономность выполнения**. По возможности процедура должна выполнять целевую задачу в полностью автономном режиме. В идеальном случае процедуры не должны иметь связей с другими внешними данными. Это означает, к примеру, что правильно запрограммированную процедуру можно скопировать в совершенно другой проект, и она будет работать в нем так же, как и в исходном. Автономность способствует многократному использованию кода и упрощает отладку приложения путем последовательной изоляции логически отличных фрагментов кода.
- **Отказ от повторяющегося кода**. При написании нетривиальных приложений Excel вы часто будете замечать, что одна и та же операция выполняется в разных местах программного кода. В подобном случае повторяющийся код следует поместить в отдельную процедуру. Таким образом, уменьшится количество операций по изменению или обновлению кода. Кроме того, общую процедуру нужно будет оптимизировать только один раз, что повышает надежность всего приложения. Все это приводит к повышению качества кода, хотя служит и другой немаловажной цели — частому повторному использованию кода. При изолировании общих операций в виде отдельных процедур вы заметите, что можете часто применять эти процедуры в других приложениях. Чем логичнее вы структурируете все выполняемые операции, тем меньше времени вам потребуется на разработку нового приложения на основе протестированных ранее процедур, сохраненных в виде библиотеки.
- **Изоляция комплексных операций**. Во многих реальных приложениях вы будете сталкиваться с ситуациями, в которых отдельные фрагменты программного кода слишком сложны и специфичны, чтобы использовать их в других приложениях. Эти разделы программного кода нужно изолировать в виде отдельных процедур, что упростит их дальнейшую отладку и поддержку.
- **Уменьшение размера процедуры**. Слишком объемные процедуры сложны для понимания, отладки и поддержки даже для опытных программистов, которые занимаются их написанием. Если ваша процедура содержит больше 150–200 строк кода, то, скорее всего, она предназначена для выполнения множества задач, а поэтому ее нужно разбить на множество процедур, выполняющих только по одной задаче.
- **Ограничение количества аргументов процедур**. Чем больше аргументов принимает процедура, тем сложнее она для понимания и тем менее эффективно выполняется. В принципе, вы должны ограничить количество аргументов процедуры пятью и меньше. Это, правда, не означает замену аргументов процедур общедоступными переменными или переменными уровня модуля. Если в исходной процедуре применяется больше пяти аргументов, то, возможно, логику процедуры или даже всего приложения следует пересмотреть.

Основные методы разработки приложений

В этом разделе описаны лучшие методы разработки, применяемые при создании любых приложений. В других главах этой книги рассматриваются методы разработки, наиболее эффективные с точки зрения приложений, рассматриваемых в данных главах.

Комментирование кода

Правильное комментирование кода является одной из самых важных операций, выполняемых при разработке приложений Excel. Ваши комментарии к программному коду должны четко и в полной мере описывать его структуру, принципы использования каждого объекта и процедуры, а также назначение кода. Комментарии также позволяют отслеживать изменения в коде, о чем рассказывается далее в этой главе.

Комментарии в программном коде важны как для его создателя, так и для других разработчиков, которым придется работать с данным кодом. Польза от комментариев в коде для других разработчиков вполне очевидна. Если вам пока не хватает опыта для эффективного выполнения поставленных задач, то после возвращения к исходным разработкам комментарии существенно упростят их анализ. Довольно часто разработчики пишут исходную версию приложения, а затем, спустя длительное время, корректируют ее и обновляют. Вы будете удивлены тем, насколько запутанно будет выглядеть написанный другими разработчиками или даже вами код после повторного ознакомления с ним спустя определенное время. Комментарии в коде помогут вам детально разобраться в его особенностях.

Комментарии должны добавляться на всех трех основных уровнях приложения: модуля, процедуры и отдельных разделов или фрагментов кода. Ниже приведены комментарии для каждого уровня приложения.

Комментарии уровня модуля

Если вы используете соглашения об именовании модулей, описанные ранее в этой главе, то любой разработчик, просматривающий ваш код, будет иметь примерное представление о назначении кода, содержащегося в каждом модуле. Вам нужно добавить короткий комментарий вверху каждого модуля с подробным описанием назначения модуля.

Примечание. При рассмотрении комментариев в программном коде обобщенным термином *модуль* называются стандартные модули, модули классов и модули кода пользовательских форм.

Корректный комментарий уровня модуля помещается вверху кода модуля и выглядит примерно так, как показано в листинге 3.2.

Листинг 3.2. Пример комментария уровня модуля

```
'
' Описание:  Краткое описание назначения кода в данном модуле
'
Option Explicit
```

Комментарии уровня процедуры

Комментарии уровня процедуры в коде приложения, как правило, более подробны. В блоке комментария уровня процедуры вы описываете назначение процедуры и приводите инструкции по ее использованию, указываете все аргументы и их назначение, а также указываете возвращаемые значения в случае функции.

Комментарии уровня процедуры могут также служить для примитивного отслеживания изменений, позволяя указывать в коде дату и описание правок, внесенных в процедуру. Корректный комментарий уровня процедуры, как показано в листинге 3.3, помещается над верхней строкой кода процедуры. Комментарий из листинга 3.3 приведен для функции. Единственное различие между блоком комментария для функции и блоком комментария для обычной процедуры состоит в том, что блок комментария для процедуры не содержит раздела `Возвращаемое значение`, поскольку процедуры значение не возвращают.

Листинг 3.3. Пример комментария уровня процедуры

```
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' Комментарии: Размещает изменяемую диаграмму или
' предлагает пользователю выбрать диаграмму из множества
' уже размещенных.
'
' Аргументы: chtChart        Эта функция возвращает
'                            ссылку на объект изменяемой
'                            диаграммы или значение
'                            Nothing в случае ее
'                            отмены пользователем.
'
' Возвращаемые значения:     True -- в случае успешного выполнения
'                            или False -- в случае ошибки или отмены
'                            пользователем.
'
' Дата     Разработчик    Действие
' ------------------------------------------------------------
' 07/04/02  Роб Боуви     Создана
' 10/14/03  Роб Боуви     Получение ошибки для диаграмм
'                         без рядов данных
' 11/18/03  Роб Боуви     Получение ошибки при неактивной
'                         рабочей книге
```

Внутренние комментарии

Внутренние комментарии представляют собой комментарии, которые добавляются в теле фрагментов кода. Эти комментарии используются для описания назначения любой части кода, что не является очевидным даже при детальном изучении. Внутренние комментарии описывают *назначение* кода, а не выполняемые им *операции*. К слову сказать, разница между назначением и выполняемыми действиями не всегда очевидна. В листингах 3.4 и 3.5 показаны два примера одного и того же кода: один с неправильным комментарием, а другой — с правильным.

Листинг 3.4. Пример неправильного внутреннего комментария к коду

```
' Циклический просмотр массива asInputFiles.
For lIndex = Lbound(asInputFiles) To Ubound(asInputFiles)
   '...
Next lIndex
```

Комментарий в листинге 3.4 абсолютно бесполезен. Во-первых, он описывает только одну строку кода, расположенную непосредственно под ним, не объясняя назначения всего цикла. Во-вторых, комментарий просто указывает на действие следующей строки кода. Эта информация очевидна при одном лишь взгляде на эту строку кода. Если вы удалите комментарий, показанный в листинге 3.4, то ничего важного не потеряете.

Листинг 3.5. Пример правильного внутреннего комментария к коду

```
' Импорт определенного списка исходных файлов в рабочую область
' рабочего листа с данными.
For lIndex = Lbound(asInputFiles) To Ubound(asInputFiles)
   '...
Next lIndex
```

В листинге 3.5 приведен комментарий, добавляющий в код много полезной информации. В нем описывается не только назначение кода, но также полная структура цикла. После прочтения этого комментария вы будете знать, какие операции выполняются в коде цикла.

В инструкциях по написанию внутренних комментариев встречаются и исключения. Самое важное исключение касается комментариев, используемых для описания структуры элементов управления. Выражения `If...Then` и `Do...Loops` по мере их разрастания могут существенно усложнить понимание кода. Суть в том, что когда эти структуры становятся большими и громоздкими, вы не можете видеть весь их код в отдельном окне. Таким образом, очень сложно запомнить все применяемые для управления элементом управления выражения. Например, при оценке очень длинной процедуры вы часто будете наблюдать фрагмент кода, подобный показанному в листинге 3.6.

Листинг 3.6. Непонятные фрагменты кода управления элементами управления

```
     End If
     lNumInputFiles = iNumInputFiles - 1
   Loop
End If
```

Какие логические проверки выполняются в листинге 3.6 с помощью двух структур `If...Then` и какое выражение управляет циклом `Do...While`? После включения в эти программные структуры больших фрагментов кода вы просто не сможете ничего узнать о выполняемых ими операциях без прокрутки кода процедуры в окне, поскольку весь блок кода не будет одновременно помещаться на экране. В качестве упрощенного решения этой проблемы вы можете воспользоваться комментариями в конце блока кода для элемента управления, как показано в листинге 3.7.

Листинг 3.7. Программные структуры для обработки элементов управления

```
    End If   '   If   bContentsValid   Then
    lNumInputFiles = lNumInputFiles - 1
  Loop       '   Do While lNumInputFiles > 0
End If       '   If bInputFilesFound   Then
```

Хотя текст комментариев из листинга 3.7 отличается от кода вверху каждой структуры для обработки элементов управления, при прочтении комментариев вы сразу понимаете, о чем идет речь. Комментарии такого типа должны использоваться везде, где применяются программные структуры, слишком большие для одновременного представления в окне кода.

Как избежать наихудшей ошибки, связанной с комментированием кода

Как бы это ни казалось очевидным, но наиболее распространенная и серьезная ошибка, связанная с комментированием кода, заключается в том, что разработчики не обновляют комментарии при изменении кода. Нам доводилось видеть много проектов, в которых, на первый взгляд, используются хорошо продуманные комментарии. Но в результате глубокого анализа оказывалось, что все комментарии созданы для старой версии проекта и не имеют никакого отношения к текущему коду.

При попытке понять проект лучше вообще не изучать комментарии, чем изучать неправильные. Плохие комментарии вводят в заблуждение. Всегда храните комментарии в соответствии с текущей версией. Старые комментарии можно либо удалять, либо хранить в виде набора записей для отслеживания изменений. Мы рекомендуем удалять устаревшие строчные комментарии, поскольку они внесут путаницу в код, вызывая сложности в понимании по причине большого количества строк с бесполезными комментариями. Используйте комментарии уровня процедур в качестве механизма отслеживания изменений там, где это необходимо.

Структурирование кода

Под структурированием подразумевается правильность организации кода. Правильно структурированный код делает логическую структуру программы максимально простой для изучения. Это очень важный момент. Для компьютера структурирование кода не играет большой роли. Единственная причина структурирования данных в программном коде — это улучшение его восприятия человеком. Как и в случае с соглашениями об именовании, безоговорочное следование правилам соглашения о структурировании делает код самозадокументированным. Основные элементы выделения в коде структурных элементов — это отступы и интервалы. К интервалам относят символы пробела, табуляции и пустые строки. Далее в этом разделе описываются способы использования интервалов и отступов для создания корректно структурированного кода.

Группируйте связанные элементы кода и отделяйте несвязанные элементы кода с помощью пустых строк. Пустые строки, разделяющие фрагменты кода в процедуре, выполняют ту же функцию, что и межабзацные интервалы в тексте книги. Они помогают определить в длинном коде отдельные логические блоки. В листинге 3.8 показан пример того, как с помощью пустых строк можно повысить удобочитаемость кода. Даже без изучения комментариев вам будет вполне понятно, какие из строки объединены в отдельные логические блоки.

Листинг 3.8. Использование пустых строк для выделения в коде структурных блоков

```
'  Восстановление свойств объекта Application
Application.ScreenUpdating = True
Application.DisplayAlerts = True
Application.EnableEvents = True
Application.StatusBar = False
Application.Caption = Empty
Application.EnableCancelKey = xlInterrupt
Application.Cursor = xlDefault

'  Удаление всех пользовательских панелей инструментов
For Each cbrBar In Application.CommandBars
  If Not cbrBar.BuiltIn  Then
    cbrBar.Delete
  Else
    cbrBar.Enabled = True
  End If
Next cbrBar
'  Восстановление строки меню рабочего листа
With Application.CommandBars(1)
  .Reset
  .Enabled = True
  .Visible = True
End With
```

Для строк кода в пределах отдельного блока задаются одинаковые отступы. С помощью отступов очень просто представить логическую структуру всего кода. В листинге 3.9 показан небольшой фрагмент кода из листинга 3.8, в котором структура программы представляется исключительно с помощью отступов. Достаточно всего лишь бегло просмотреть этот фрагмент кода, чтобы понять его логическую структуру и распознать отдельные блоки.

Листинг 3.9. Правильное использование выравнивания и отступов

```
  '  Удаление всех пользовательских панелей инструментов
  For Each cbrBar In Application.CommandBars
    If Not cbrBar.BuiltIn  Then
      cbrBar.Delete
    Else
      cbrBar.Enabled = True
    End If
  Next cbrBar
```

Символ продолжения строки применяется для того, чтобы упростить понимание длинных выражений и операторов. Помните о том, что разделение одной строки кода на несколько выполняется исключительно для отображения всего кода программы без горизонтальной прокрутки окна кода. Эта операция не относится к правилам хорошего программирования и часто вызывает путаницу в коде. В листинге 3.10 показаны примеры правильного разделения строк кода.

Листинг 3.10. Правильное разделение строк кода

```
' Сложные выражения легче понять
' при правильном разбиении строки
If (uData.lMaxLocationLevel > 1) Or _
   uData.bHasClientSubsets Or _
   (uData.uDemandType = bcDemandTypeCalculate)   Then

End If

' После разделения строки длинные объявления
' функций API становятся понятнее
Declare Function SHGetSpecialFolderPath Lib "Shell32.dll" _
   (ByVal hwndOwner As Long, _
   ByRef szBuffer As String, _
   ByVal lFolder As Long, _
   ByVal bCreate As Long) As Long
```

Методы программирования на VBA

Основные методы программирования на VBA

Использование операторов уровня модуля

- **`Option Explicit`**. Обязательно используйте в каждом модуле оператор `Option Explicit`. Его важность трудно переоценить. Без оператора `Option Explicit` любая опечатка в имени переменной, сделанная вами в коде, будет приводить к автоматическому созданию новой переменной `Variant`. Ошибки этого типа очень опасны, поскольку не всегда распознаются на этапе отладки, а потому о них, как правило, извещают конечные пользователи. Однако в большинстве случаев данные ошибки приводят к получению в приложении неправильных результатов. Заметьте, что сообщения о таких ошибках поступают к разработчику только после распространения приложения. Кроме того, выполнять отладку приложения с такого типа ошибками очень непросто.

  Оператор `Option Explicit` вынуждает явным образом объявлять все переменные, которые используются в коде. Он указывает VBA выводить сообщение об ошибке компиляции (команда Debug⇨Compile (Отладка⇨Компилировать) в окне VBE) всякий раз при нахождении неопределенного ранее идентификатора. Это позволяет быстро и безболезненно находить и исправлять опечатки в коде. Для автоматического добавления оператора `Option Explicit` в начале кода каждого модуля выполните команду Tools⇨Options⇨Editor (Сервис⇨Параметры⇨Редактор) окна VBE и установите флажок Require Variable Declaration (Обязательное объявление переменных). Настоятельно рекомендуется не пренебрегать этой настройкой.

- **`Option Private Module`**. Оператор `Option Private Module` делает все процедуры в пределах модуля, в котором они используются, недоступными из пользовательского интерфейса Excel или из других проектов Excel. Используйте этот оператор для сокрытия процедур, которые не должны вызываться извне вашего приложения.

Примечание. Метод `Application.Run` позволяет обойти ограничение, заданное оператором `Option Private Module`, и выполнить локальные процедуры из модулей, в коде которых введен этот оператор.

- **`Option Base 1`**. Оператор `Option Base 1` указывает применять во всех переменных массивов, для которых не задана нижняя граница, начальный индекс, равный единице. В большинстве случаев не рекомендуется применять оператор `Option Base 1`. Всегда явно определяйте верхние и нижние границы для каждой переменной массива, используемой в коде. Процедура, созданная в модуле, в котором введен оператор `Option Base 1`, может работать некорректно после копирования в модуль, в котором этот оператор не применяется. Подобное ограничение на нумерацию элементов массива противоречит таким важнейшим принципам разработки процедур, как возможность их повторного использования.
- **`Option Compare Text`**. Оператор `Option Compare Text` указывает сравнивать строки в пределах модуля, в котором он введен, как текстовые, а не двоичные значения. При сравнении строк как текстовых значений верхний и нижний регистры одного и того же символа интерпретируются, как идентичные, в то время как при сравнении двоичных значений они отличаются. Применения оператора `Option Compare Text` следует избегать по тем же причинам, что и оператора `Option Base 1`. После их добавления в модуль процедуры изменяют свое поведение. Сравнение текстовых строк выполняется гораздо сложнее, чем сравнение двоичных значений. Поэтому оператор `Option Compare Text` замедляет все операции, требующие сравнения строк, в текущем модуле. Большинство функций сравнения строк в Excel и VBA имеют аргумент, с помощью которого определяется способ сравнения: в виде двоичных или текстовых значений. С помощью этого аргумента можно выполнять сравнение текстовых значений только в тех местах кода, где это действительно необходимо.

  В некоторых ситуациях все же требуется использовать оператор `Option Compare Text`. Чаще всего они возникают при сравнении строковых значений с помощью оператора `Like`, но без учета регистра символов. Единственный способ указать оператору `Like` выполнить сравнение без учета регистра заключается в добавлении в код модуля оператора `Option Compare Text`. В этом случае вы должны изолировать процедуры, которые принимают этот оператор, в отдельном модуле кода, чтобы он не повлиял на ход выполнения других процедур, не требующих использования этой опции. Не забудьте указать в специальном комментарии уровня модуля, почему применяется этот оператор.

Переменные и константы

Избегайте повторного использования переменных. Каждая переменная, объявленная в программе, должна служить только одной цели. Использование одной и той же переменной для выполнения различных задач требует добавления в код всего лишь одной строки с ее объявлением, но приводит к большой неразберихе в программе. Если вы пытаетесь определить, как работает процедура, зная, для чего применяется каждая переменная в определенном фрагменте кода, то, естественно, будете предполагать, что в других фрагментах кода эта переменная будет выполнять такие же задачи. Если это не так, то логику программы вам понять будет очень непросто.

Избегайте типа данных `Variant`. По возможности не используйте тип данных `Variant`. К сожалению, VBA не относится к строгим языкам программирования. Поэтому вы можете объявлять переменные без указания их типа данных. В результате VBA назначит этим переменным тип данных `Variant`. Ниже приведены главные причины, по которым не стоит использовать тип данных `Variant`.

- **Тип данных `Variant` малоэффективен**. Внутренне тип данных `Variant` имеет очень сложную структуру, включая в себя любой другой тип данных в VBA. К значениям типа `Variant` нельзя получить доступ и модифицировать их непосредственно, как значения таких фундаментальных типов данных, как `Long` и `Double`. Таким образом, если VBA требуется выполнить любую операцию с типом данных `Variant`, то в фоновом режиме при этом вызываются многие API-функции.
- **Значения, хранящиеся в типе данных `Variant`, могут вести себя непредсказуемо**. Поскольку переменная типа `Variant` предназначена для хранения любого значения, исходящий тип данных, получаемый при извлечении значения, не всегда будет совпадать с входящим типом, который имели данные перед сохранением в значении типа `Variant`. При доступе к данным типа `Variant` VBA будет принудительно присваивать данным тип, с его точки зрения наиболее подходящий для сохраняемого значения. Поэтому при использовании переменных типа `Variant` всегда явным образом указывайте нужный тип данных при извлечении их значений.

Остерегайтесь некорректного преобразования типов данных. Некорректное преобразование типов данных представляет собой еще один недостаток нестрогого языка программирования VBA. Данный тип ошибок возникает, когда VBA автоматически преобразует один тип данных в другой, совершенно не связанный с исходными данными. В качестве самого распространенного примера можно рассмотреть преобразование типа данных `String`, который содержит числа, в тип данных `Integer` или `Boolean` и обратное преобразование полученного результата в тип `String`. Не стоит совмещать в коде переменные с различными типами данных без использования функций явного преобразования типа (`CStr`, `CLng`, `CDbl` и т.д.), которые специально предназначены для указания правильного метода интерпретации их значений.

Избегайте объявлений типа `As New`. Никогда не объявляйте переменные объекта с помощью выражения `As New`. Например, следующую форму объявления переменной объекта использовать нельзя.

```
Dim rsData As New ADODB.Recordset
```

Когда VBA сталкивается со строкой кода, в которой используется эта переменная, и она не инициализирована, то в коде автоматически создается новый экземпляр переменной. Этот экземпляр *никогда* не будет себя вести так, как требуется. Хорошим стилем в программировании считается обеспечение полного контроля за созданием любых объектов, используемых в программе. Если VBA встречает в коде неинициализированную переменную объекта, то чаще всего возвращает ошибку, о которой вам нужно узнать как можно скорее. Поэтому правильно переменная объекта из предыдущего примера должна объявляться и инициализироваться так.

```
Dim rsData As ADODB.Recordset
Set rsData = New ADODB.Recordset
```

При таком объявлении и инициализации переменной можете не опасаться ее удаления в процедуре. Если вы впоследствии случайно обратитесь к ней (после удаления), то

VBA немедленно возвратит ошибку выполнения “Object variable or `With` block variable not set” (Объектная переменная или переменная блока `With` не установлена).

Всегда уточняйте полные имена объектов. Всегда в объявлениях переменных и в коде вводите полные имена объектов, содержащие префикс имени класса. Данное требование вызвано тем, что во многих библиотеках объектов используются объекты с одинаковыми именами. Если вы объявите переменную под простым именем объекта, и данное имя представляет большое количество объектов в самых разных библиотеках, то VBA добавит в список Tools⇨References (Сервис⇨Ссылки) ссылку на объект из первой библиотеки, в которой это имя встречается. Чаще всего оно представляет далеко не тот объект, которого вы ожидали.

Элементы управления пользовательских форм представляют самый распространенный источник переменных объектов, которые объявляются не под полными именами, что приводит к возникновению частых ошибок. Например, переменную объекта для элемента управления `TextBox` текущей пользовательской формы можно объявить следующим образом.

```
Dim txtBox As TextBox
Set txtBox = Me.TextBox1
```

К сожалению, как только VBA попытается выполнить вторую строку кода, будет возвращено сообщение об ошибке “Type mismatch” (Несоответствие типов). Причина ошибки в том, что объект `TextBox` содержится и в библиотеке объектов Excel, которая в списке ссылок Tools⇨References находится перед библиотекой объектов `MSForms`. Таким образом, правильный код должен выглядеть так.

```
Dim txtBox As MSForms.TextBox
Set txtBox = Me.TextBox1
```

Никогда не определяйте строгие границы массива. Если в коде используется переменная массива, которая запрашивается в циклах, никогда не задавайте границы массива строго. Вместо этого для определения границ массива используйте функции `LBound` и `Ubound`, как показано в листинге 3.11.

Листинг 3.11. Корректный способ циклической обработки значений массива

```
Dim lIndex As Long
Dim alListItems(1 To 10) As Long

' Операторы загрузки массива

For lIndex = LBound(alListItems) To Ubound(alListItems)
   ' Обработка каждого значения
Next lIndex
```

Дело в том, что по ходу создания и исправления приложения границы массива часто изменяются. Если в цикле границы массива заданы однозначно, например 1 и 10, то вам придется обновлять цикл при каждом изменении границ массива `alListItems`. В противном случае в коде приложения будут возникать ошибки. Функции `LBound` и `UBound` применяются для автоматического изменения цикла после обновления размера массива.

Всегда устанавливайте счетчик после ключевого слова `Next`. В листинге 3.11 продемонстрирован еще один принцип хорошего программирования. Всегда после ключевого

слова Next в циклической структуре необходимо определять счетчик. Хотя VBA не обязует к этому, программный код будет более понятен, если в цикле явно объявлять счетчик, особенно в случае ввода между ключевыми словами For и Next множества строк.

Используйте константы. Константы — это очень полезные элементы любой программы. Они служат многим целям, включая следующие.

- Устраняют "безымянные" числа, снабжая их описательными названиями. Например, попробуйте догадаться, что означает число 50 в следующей строке.

  ```
  If lIndex < 50 Then
  ```

  Если сразу после написания кода вы не помните, что представляет собой число 50, то при повторном его изучении вы тем более не сможете это вспомнить. Если воспользоваться следующим кодом, то впоследствии будет понятно, какая проверка выполняется с помощью конструкции If...Then.

  ```
  Const lMAX_NUM_INPUT_FILES As Long = 50
  ' Программный код

  If lIndex < lMAX_NUM_INPUT_FILES Then
  ```

  Если вам нужно извлечь значение константы в процессе разработки приложения, щелкните в окне VBE на имени константы правой кнопкой мыши и выберите из контекстного меню опцию Definition (Объявление). Курсор переводится в строку, в которой объявлена выбранная константа. В режиме прерывания во время отладки эта задача выполняется еще проще. Наведите указатель мыши на константу, и ее значение отобразится во всплывающем окне.

- Константы повышают эффективность кода и помогают избегать частых ошибок, предотвращая дублирование данных. Предположим, что в предыдущем примере вы ссылаетесь на максимальное допустимое количество исходных файлов в нескольких местах программы. На некотором этапе вам может потребоваться обновить программу, обеспечив в ней обработку большего количества файлов. Если максимальное количество исходных файлов задано строго, то вам придется найти все подобные места в коде и изменить число в каждом из случаев. При использовании константы вам потребуется изменить значение всего лишь один раз в объявлении константы, и оно автоматически будет применено во всех местах кода, где эта константа упоминается. Рассмотренная выше ситуация является частым источником ошибок. Ее можно избежать, если использовать константы вместо строго определенных числовых значений.

Область действия переменных

Глобальные переменные всегда потенциально опасны для использования в коде. Их можно изменить в любом месте приложения без предварительного предупреждения, что делает их значения непредсказуемыми. Они также нарушают важнейший принцип программирования: инкапсуляцию. Всегда создавайте переменные с минимально возможной областью действия. Начните с создания всех переменных с локальной (на уровне процедуры) областью действия и расширяйте область действия для переменной только тогда, когда это действительно необходимо.

В определенных ситуациях общедоступные (глобальные) переменные использовать эффективнее или просто необходимо.

- Данные перед использованием сохраняются в программном стеке. Например, если процедура A считывает некоторые данные и затем передает их процедуре B, которая, в свою очередь, передает их процедуре C, а та передает их процедуре D, в которой, наконец, эти данные обрабатываются, то лучше сделать так, чтобы данные передавались из процедуры A непосредственно в процедуру D, что проще всего реализуется с помощью глобальной переменной.
- Наследуемые общедоступные классы, такие как классы обработки событий на уровне приложения, требуют использования глобальной переменной объекта, поскольку они никогда не должны выходить за пределы области действия во время работы приложения.

Раннее и позднее связывание

Разница между ранним и поздним связыванием трактуется весьма неоднозначно, что часто вызывает неразбериху при создании объектов. *Единственное*, что влияет на принцип связывания объекта, — это способ объявления переменной, содержащей ссылку на текущий объект. Переменные, объявленные с типом данных `Object` или `Variant`, всегда имеют позднее связывание. В листинге 3.12 показан пример позднего связывания, а в листинге 3.13 — пример раннего связывания.

Листинг 3.12. Позднее связывание с объектом ADO Connection

```
Dim objConnection As Object

' Не имеет значения, как создается объект; он имеет позднее
' связывание вследствие объявления переменной оператором As Object
Set objConnection = New ADODB.Connection
Set objConnection = CreateObject("ADODB.Connection)
```

Листинг 3.13. Раннее связывание с объектом ADO Connection

```
Dim cnConnection As ADODB.Connection

' Не имеет значения, как создается объект; он имеет раннее
' связывание, поскольку в объявлении переменной указан тип данных
Set cnConnection = New ADODB.Connection
Set cnConnection = CreateObject("ADODB.Connection)
```

Чтобы воспользоваться ранним связыванием с объектами вне объектной модели Excel, вы должны создать ссылку на соответствующую библиотеку объектов с помощью команды Tools⇨References окна Visual Basic Editor. Например, для раннего связывания переменных с объектами ADO вы должны создать ссылку на библиотеку Microsoft ActiveX Data Objects2.*x* Library, где *x* представляет версию ADO, которую вы намерены использовать.

Старайтесь использовать ранее связанные переменных с объектами везде, где это возможно. Раннее связывание имеет определенные преимущества перед поздним связыванием.

- **Повышенная производительность приложения**. При использовании переменной объекта, тип данных которой становится известен во время компиляции, VBA определяет ячейки памяти, в которых сохраняются данные свойств и методов,

характерных для текущего объекта, и сохраняет их вместе с кодом. Во время выполнения VBA-кода в случае обращения к одному из свойств или методов ранее связанного объекта извлекаются данные из заранее определенной при компиляции ячейки памяти. (Процедура описана весьма упрощенно. В действительности VBA сохраняет смещение выполняемого кода относительно общей начальной точки области памяти, которая определяет начало структуры `vTable` объекта.)

При использовании позднего связывания переменной с объектом VBA ничего не знает о том, какой тип объекта будет сохраняться в переменной. Поэтому VBA не сможет оптимизировать процедуры вызова свойств или методов этих объектов во время компиляции. Это означает, что каждый раз, когда VBA во время выполнения сталкивается с задачей вызова свойства или метода объекта позднего связывания, он сначала определяет тип переменной, в которой хранить объект, извлекает имя запрашиваемого свойства или метода, находит ячейку памяти, в которой располагаются соответствующие данные, и только после этого выполняет код, размещенный по найденному адресу памяти. Эта операция более длительна, чем вызов свойств и методов при раннем связывании.

- **Строгая проверка типов данных**. Если в примере позднего связывания из листинга 3.12 вы случайно замените ссылку на объект ADO `Connection` ссылкой на объект `Command`, то VBA не возвратит сообщение об ошибке. Вы разве что впоследствии обнаружите в коде проблему выполнения несуществующих свойств и методов объекта `Connection`, которые не поддерживаются в объекте `Command`. В случае с ранним связыванием VBA немедленно обнаружит попытку назначения переменной ссылки на объект неправильного типа, а потому выведет сообщение об ошибке "Type mismatch" (Несоответствие типов). Сбои при вызове несуществующих свойств или методов возникают еще раньше, до выполнения кода. Во время компиляции VBA попытается найти имя свойства или метода, вызываемого из указанной библиотеки объектов, и возвратит ошибку, если такое имя не было обнаружено.
- **Доступность функции автозаполнения**. Раннее связывание переменных объектов также способствует упрощению программного кода. Поскольку VBA точно знает, какой тип объекта сохраняется в переменной, он заранее анализирует соответствующую библиотеку объектов и снабжает разработчика раскрывающимся списком всех свойств и методов этого объекта, избавляя от необходимости каждый раз вручную вводить в коде их имена.

Как вы догадываетесь, в некоторых случаях эффективнее использовать позднее связывание вместо раннего. Существуют две основные причины использования позднего связывания вместо раннего.

- Новая версия библиотеки объектов приложения не совместима с ранней версией этой библиотеки.

  Весьма распространенная ситуация. Если вы создаете в приложении ссылку на позднюю версию библиотеки объектов, а затем пытаетесь запустить приложение на компьютере, на котором установлена более ранняя версия этой же библиотеки, то получите сообщение об ошибке "Can't find project or library" (Невозможно найти проект или библиотеку), а перед самой ссылкой будет добавлен префикс `MISSING` (Недоступна). Самое странное в этой ошибке состоит в том, что строка кода, помеченная как источник ошибки, часто никак не связана с библиотекой объектов, вызвавшей ошибку.

Если вам все же нужно использовать объекты из приложения, в котором проявляется такая проблема, поскольку приложение будет распространяться среди пользователей с любой версией библиотеки объектов, то обратитесь к позднему связыванию всех переменных, ссылающихся на объекты исходного приложения. В случае создания новых объектов используйте функцию `CreateObject` с независимым от версии идентификатором `ProgID` объекта, а не синтаксис `= New ObjectName`.

- Вы планируете использовать приложение, которое не установлено в компьютере пользователя, и вы не можете установить его самостоятельно.

  В данном случае вам нужно использовать позднее связывание, чтобы избежать ошибок в процессе компиляции, которые немедленно возникают при попытке запуска приложения, обращающегося к несуществующей в компьютере пользователя библиотеке объектов. При позднем связывании приложение сначала проверяет существование библиотеки объектов, а затем корректно завершает свою работу, если эта библиотека в целевом компьютере не найдена.

Примечание. Даже если вы остановитесь на методе позднего связывания, имейте в виду, что раннее связывание обеспечивает настолько большой прирост производительности, что в процессе отладки и исправления приложения все же лучше применять именно его. Приведите код приложения к позднему связыванию только на конечном этапе разработки приложения, непосредственно перед последним тестированием и распространением.

Код защиты данных

Создание защитного кода относится к методам программирования, предназначенным для предотвращения возникновения ошибок в процессе коррекции кода.

Создавайте приложение для самой ранней версии Excel, в которой оно может запускаться

Несмотря на то что разработчики последних версий Excel проделали громадную работу по обеспечению совместимости с более ранними версиями программы, между разными версиями Excel все еще наблюдается множество различий. Если вы хорошо знакомы только с поздней версией Excel, то можете легко создать приложение, которое не будет запускаться в более ранней версии программы, поскольку некоторые используемые вами возможности в ней не будут поддерживаться.

Решение проблемы заключается в том, чтобы всегда разрабатывать приложения в расчете на самую раннюю версию Excel, в которой они могут запускаться. При этом вам, возможно, придется установить на одном компьютере несколько разных версий программы Excel, а лучше получить доступ к разным версиям программы на совершенно разных компьютерах. В любом случае это очень важное требование. Если вы разработали приложение в Excel 2000, отправили его пользователю Excel 97 и обнаружили, что оно не запускается, то вам придется отладить приложение и удалить в нем код, который не выполняется в Excel 97. Вы сэкономите много времени и сил, если изначально разработаете приложение в расчете на выполнение в Excel 97.

Явное объявление аргументов: `ByRef` и `ByVal`

Если в процедуре используются аргументы, то их можно объявить с помощью ключевых слов `ByRef` и `ByVal`.

- **ByRef**. Подразумевает передачу в процедуру адреса в памяти, а не значения переменной. Если вызванная процедура изменяет аргумент типа ByRef, то это изменение фиксируется в вызывающей процедуре.
- **ByVal**. Подразумевает передачу процедуре значения. Процедура может вносить изменения в аргумент ByVal, но эти изменения не будут видны в вызывающей процедуре. По сути, вызываемая процедура использует аргументы типа ByVal так, как если бы они объявлялись как локальные.

Всегда явно указывайте тип аргументов процедур: ByRef или ByVal. Если этого не сделать, то все создаваемые аргументы по умолчанию будут иметь тип ByVal. Аргументы ByVal применяются в вызываемых процедурах, если вызывающую процедуру не нужно извещать об изменениях в этих аргументах. Объявление аргументов как имеющих тип ByVal позволяет предотвратить внесенные в них изменения при передаче обратно в вызывающую процедуру.

Исключения встречаются только тогда, когда вы передаете в процедуру длинные строки (очень длинные строки), которые гораздо эффективнее управляются с помощью аргументов ByRef, или же когда аргумент процедуры имеет такой тип данных, как массив, который нельзя передать с помощью аргумента ByVal. Вам следует знать, что объявление в процедуре аргументов ByVal увеличивает вероятность возникновения ошибок несоответствия типов данных. Аргумент ByRef *должен* передаваться с тем же типом данных, с которым он был объявлен. В противном случае возникнет ошибка компиляции. В то же время VBA будет пытаться преобразовать значение, переданное аргументу ByVal, в совместимый тип данных.

Явный вызов свойства по умолчанию объекта

За возможным исключением свойства Item объекта Collection лучше не вызывать неявно свойство по умолчанию объекта, ограничившись вводом в ссылке одного только имени объекта. В листинге 3.14 показаны правильный и неправильный способы доступа к свойству объекта по умолчанию на примере элемента управления MSForms.TextBox (свойство Text является свойством по умолчанию для элемента управления MSForms.TextBox).

Листинг 3.14. Свойства, заданные по умолчанию

```
' Правильный способ
txtUsername.Text = "My name"

' Неправильный способ
txtUsername = "My name"
```

Избежав неявного использования свойств по умолчанию, вы сделаете код более понятным и защитите его от потенциальных ошибок, вызванных изменением поведения объекта по умолчанию в более поздней версии Excel или VBA.

Проверка аргументов перед их использованием в процедурах

Если процедура принимает аргументы, которые можно проверить с помощью определенных свойств (например, они должны задаваться в специфическом диапазоне значений), то перед их использованием в процедуре убедитесь в корректности переданных в аргументы значений. Ваша задача заключается в том, чтобы как можно быстрее перехватывать неправильные передаваемые в процедуру данные, сгенерировать понятное сообщение об ошибке и максимально упростить отладку кода.

По возможности для проверки правильности поведения процедур создавайте специальные средства тестирования. Средство тестирования представляет собой процедуру-оболочку, которая вызывает любую другую процедуру, многократно тестирует ее, передавая ей широкий диапазон аргументов, и сверяет полученный результат с диапазоном допустимых выходных значений. В главе 16 средства тестирования описаны более подробно.

Использование счетчиков для предотвращения образования бесконечных циклов

Создавайте циклы таким образом, чтобы в них автоматически предотвращалась попытка образования бесконечных циклов. Одна из самых распространенных причин образования бесконечных циклов заключается в использовании циклов `Do...While` и `While...Wend` для обработки ситуаций, в которых условие выхода из цикла никогда не выполняется. Это приводит к тому, что цикл продолжает выполняться бесконечно (прерывается только с помощью комбинации клавиш <Ctrl+Break> (если повезет) или диспетчера задач Windows, применяемого для завершения работы любой программы). Старайтесь всегда добавлять в цикл счетчик, который автоматически останавливает выполнение цикла, когда количество итераций достигает максимально возможного числа, которое определяется исходя из практических соображений. В листинге 3.15 показан цикл `Do...While` со встроенной защитой от бесконечных циклов.

Листинг 3.15. Защита от бесконечных циклов

```
Dim bContinueLoop As Boolean
Dim lCount As Long

bContinueLoop = True
lCount = 1

Do

   ' Приведенный здесь код должен выполняться до тех пор,
   ' пока для переменной bContinueLoop не будет задано
   ' значение False.

   ' Счетчик защиты от бесконечных циклов прерывает
   ' цикл после выполнения 10000 итераций
   lCount = lCount + 1
   If lCount > 10000 Then Exit Do

Loop While bContinueLoop
```

Единственное назначение переменной `lCount` в теле цикла состоит в прерывании цикл, если по истечении 10000 итераций условие выхода из цикла так и не будет выполнено. (Количество итераций зависит от условий решаемой задачи, а потому разное в каждом конкретном случае.) Такая структура цикла вызывает незначительное снижение производительности приложения, но при этом предоставляет защиту от бесконечных циклов. Как только вы удостоверитесь в правильности функционирования кода в теле цикла, смело удалите строки с защитой от бесконечного цикла или просто закомментируйте их.

Команда Debug⇨Compile

Никогда не оставляйте код непроверенным после внесения даже самых незначительных изменений. Он должен безукоризненно выполняться с помощью команды Debug⇨Compile (Отладка⇨Компилировать). Регулярно забывая выполнять ее, вы гарантируете себе бессонные ночи, проведенные за отладкой приложения.

Использование имен для ссылки на объекты рабочих листов

Всегда ссылайтесь на рабочие листы и листы диаграмм в приложении, используя в коде их имена. Идентификация рабочих листов в коде по названиям их ярлыков чревата ошибками, поскольку конечные пользователи могут легко изменять эти названия, "нарушая" код, в котором они используются.

Проверка типа данных выделения

Если вы пишете процедуру, предназначенную для управления выделенным пользователем объектом специального типа, всегда проверяйте тип выделенного объекта, используя функцию `TypeName` или конструкцию `If TypeOf...Is`. Например, если в процедуре нужно обработать диапазон ячеек, выделенных пользователем, убедитесь, что это выделение представлено объектом `Range`, как показано в листинге 3.16.

Листинг 3.16. Проверка правильности типа выбранного объекта

```
' Код, предназначенный для обработки диапазона
If TypeOf Selection Is Excel.Range Then
   ' Все верно, это объект Range
   ' Продолжение кода обработки диапазона
Else
   ' Ошибка, это не объект Range
   MsgBox "Please select a range.", vbCritical, "Error!"
End If
```

Управление изменениями

Управление изменениями, известное также как управление версиями, заключается в выполнении двух основных задач: поддержке набора предыдущих версий приложения, которые можно использовать для восстановления после возникновения различных программных или системных ошибок, и внесении в приложение запланированных изменений.

Сохранение версий

Когда большинство профессиональных разработчиков рассказывают об управлении версиями, они упоминают о таком программном обеспечении, как, например, Microsoft Visual Source Safe. Однако программы подобного рода стоят весьма дорого, требуют специальных навыков, а также плохо совмещаются с приложениями, предназначенными для выполнения в Excel. Основная причина всех бед в том, что Excel не сохраняет модули кода в виде текстовых файлов. Предлагаемый нами метод управления версиями очень быстрый, простой в реализации, не требует применения специального программного обеспечения и обладает всеми преимуществами традиционной системы управления версии приложения.

Главная цель системы управления версиями — предоставить возможность восстанавливать проект до состояния более ранней версии в случае возникновения серьезных проблем с текущей версией проекта. Если вы внесли неправильные изменения в код или нечаянно повредили файл проекта, то без резервной копии, на основе которой выполняется восстановление данных, вам не удастся быстро "вернуть проект к жизни".

Простая система управления версиями, которая призвана помочь вам избежать подобных проблем, реализуется следующим способом. Сначала в папке, в которой хранится проект, создайте вложенную папку и назовите ее `Backup`. Каждый раз перед внесением серьезных изменений в проект, или не реже одного раза в день, используйте утилиту архивации данных (например, WinZip), чтобы сжать все файлы из папки проекта в файл `Backup_YYYYMMDDHH.zip`, где `YYYY` — год, `MM` — месяц, `DD` — день, `HH` — час. Таким образом вы снабжаете резервную копию уникальным именем, указывающим дату ее создания. Более того, Windows будет сортировать файлы резервных копий в правильном порядке при просмотре в программе Проводник. Переместите полученный файл архива в папку `Backup` и продолжайте работу над текущей версией проекта.

Если у вас возникала проблема, восстановите проект из последней резервной копии. Вы, конечно же, потеряете небольшую часть последних изменений, но при регулярном сохранении резервных копий проекта потери сводятся к минимуму. Как только вы будете уверены, что получили стабильную и полностью отлаженную версию проекта, удалите большинство файлов промежуточных резервных копий из папки `Backup`. Достаточно сохранять резервные копии отлаженных версий проекта хотя бы раз в неделю.

Документирование изменений с помощью комментариев

При обновлении кода, когда в его структуру вносятся серьезные изменения, вы должны снабжать процедуры приложения комментариями, которые содержат описание правок, дату изменения кода и имя автора изменений (см. листинг 3.3). Все нетривиальные правки, вносимые в код, также должны обозначаться с помощью комментариев, в которых указываются дата изменения и имя разработчика, который вносил правки (если приложение разрабатывается совместно).

Резюме

При использовании соглашения об именовании элементов приложения, предложенных в этой главе или разработанных самостоятельно, применяйте его во всех без исключения приложениях. Только тогда код можно считать самозадокументированным и простым для понимания. Разделите приложение на различные логические уровни, представив их независимыми модулями. Это предотвратит реконструкцию всего приложения при внесении изменений только в один из модулей. Комментируйте код приложения на всех доступных уровнях. Понять любой фрагмент кода будет гораздо легче, если его назначение детально описано в комментариях. Остальные методы правильного программирования, представленные в этой главе, помогут вам создавать устойчивые к сбоям, предельно понятные и максимально надежные приложения.