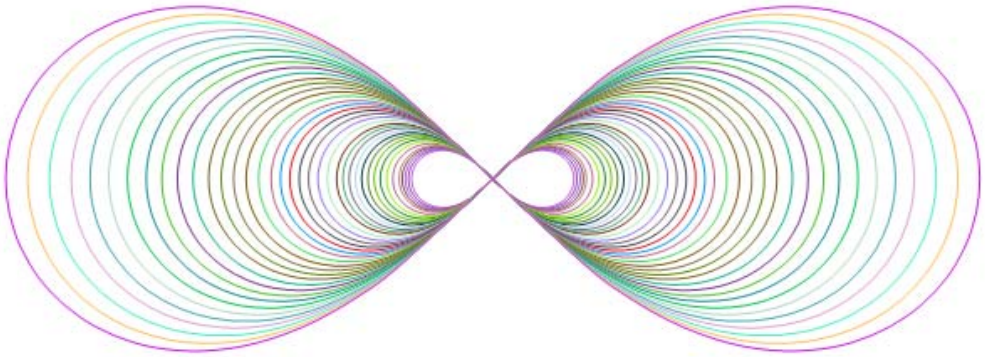
Андрей Колесов, Ольга Павлова

СОВЕТЫ
ТЕМ, КТО ПРОГРАММИРУЕТ на Visual Basic и MS Office/VBA



"КомпьютерПресс",
1996-2002

© 1996-2002, Андрей Колесов, Ольга Павлова
http://www.visual.2000.ru/develop/ms-vb/index.htm

Андрей Колесов

© 1996, Андрей Колесов
Авторский вариант. Статья была опубликована с незначительной литературной правкой в журнале "КомпьютерПресс" N 3/96, с. 62-64.

Когда в январе 1996 года я отправлял эту статью в "КомпьютерПресс", то никак не думал, что будет написан даже четвертый совет. Сегодня (май 2000 года) их число уже приблизилось к трехстам...

Совет 1. Используйте содержательные имена объектов

При создании нового объекта (формы или элемента управления) ему автоматически присваивается идентификатор (свойство Name), который состоит из имени, назначаемого по умолчанию, и порядкового номера объекта данного типа. Для элементов управления нумерация ведется в пределах каждой отдельной формы, а для формы — в рамках текущего проекта.

Однако при создании формы с достаточно большим числом объектов, особенно одного типа, гораздо удобнее вместо безликих названий наподобие Command1, Command2, Command3 использовать более содержательные имена, например, Exit, OK, Cancel.

Изменение имен объектов выполняется с помощью соответствующих установок свойства Name в окне Properties.

Если для элементов управления изменение имен является, в общем-то, необязательным делом, то для форм, используемых в различных проектах, установка уникальных идентификаторов является просто необходимой. При попытке загрузки сделанной ранее формы Form2 в новый проект, в котором уже создана форма с таким же именем, будет, естественно, выдаваться сообщение о невозможности этой операции.

Здесь следует иметь в виду одно важное обстоятельство. Дело в том, что имя элемента управления автоматически используется в формировании относящихся к нему событийных процедур, задействованных в данном модуле формы. (Для самой формы — она одна в модуле — используется фиксированное имя Form независимо от значения свойства Name.) Поэтому, создав, например, командную кнопку Command1, а затем описав для нее обработку события "щелчок мышью", вы получите процедуру с названием Command1_Click. Если же после этого вы решите изменить идентификатор кнопки, то получится так, что ее ранее задействованные событийные процедуры останутся в этом модуле со старыми названиями. Причем все эти процедуры превратятся просто в мусор, так как они никак не будут использоваться: для данной кнопки нужно будет заново описывать все процедуры или менять их старые имена на новые.

То же самое относится и к идентификаторам свойств объекта, с той лишь разницей, что при запуске программы будет сразу выдано сообщение об ошибке — "variable not defined", что в данном случае означает "неверный идентификатор свойства". Поэтому при изменении имени элемента управления рекомендую воспользоваться таким советом:

Совет 1.1. При изменении имени объекта сделайте коррекцию идентификаторов всех соответствующих ему процедур и свойств в коде модуля с использованием команды Replace в меню Edit (например, заменить "Command1" на "cmdExit").

Конечно, было бы гораздо удобнее, если бы такая модификация выполнялась автоматически при изменении имени элемента управления, — это явное упущение разработчиков языка. И в связи с этим еще одна очевидная рекомендация.

Совет 1.2. Присваивайте собственное уникальное имя элементу управления сразу после его создания и в будущем старайтесь не изменять его без особой нужды. Помните, что эти идентификаторы могут использоваться и в других модулях программы для адресации к свойствам и методам данных объектов.

Разумеется, при присвоении нового идентификатора объекта желательно отразить в имени не только его функциональное назначение, но и его тип. Для этого лучше всего использовать в именах объекта префиксы (приставки), предлагаемые разработчиками VB в качестве неформального стандарта (см. таблицу), которого стараются придерживаться все пользователи языка.

Таблица. Соглашение об именах объектов (для стандартного набора объектов)

Тип объекта	**Название объекта (свойство Name)**		
	По умолчанию	**Префикс**	**Пример**
Form	Form	Frm	frmFileOpen
Check box	Check	Chk	chkReadOnly
Combo box	Combo	Cbo	cboRussian
Command button	Command	Cmd	cmdCancel
Data	Data	Dat	datBiblio
Directory list box	Dir	Dir	dirSource
Drive list box	Drive	Drv	drvTarget
File list box	File	Fil	filSource
Frame	Frame	Fra	fraLanguage
Grid	Grid	Grd	grdPrices

Horizontal scroll bar	HScroll	Hsb	hsbVolume
Image	Image	Img	imgIcon
Label	Label	Lbl	lblHelpMessage
Line	Line	Lin	linVertical
List box	List	Lst	lstPolicyCodes
Menu	Menu	Mnu	mnuFileOpen
OLE	OLE	Ole	oleObject1
Option button	Option	Opt	optRussian
Picture box	Picture	Pic	picDiskSpace
Shape	Shape	Shp	shpCircle
Text box	Text	Txt	txtGetText
Timer	Timer	Tmr	tmrAlarm
Vertical scroll bar	VScrll	Vsb	vsbRate

Удобство применения именно префиксов заключается и в том, что список объектов будет автоматически упорядочен по типам объектов. Обратите внимание, что при написании префикса предлагается использовать только строчные (маленькие буквы) — это позволяет сразу отличить стандартные имена от присвоенных пользователем. Применение данного соглашения поможет вам не запутаться в собственных программах, и самое главное — предоставит другим пользователям ваших программ возможность легко ориентироваться в них.

Совет 2. Оптимизируйте время загрузки нового проекта

При создании нового проекта (команда New File) в качестве начального списка дополнительных элементов управления (VBX), включаемых в окно ToolBox, используется файл AUTOLOAD.MAK (он хранится в главном каталоге самого пакета VB). Эта операция выполняется автоматически каждый раз при запуске среды VB.

При инсталляции VB в файл AUTOLOAD.MAK записываются все дополнительные VBX'ы, входящие в состав данной редакции. Для VB 3.0 Professional их число равно четырнадцати, и время их загрузки даже для 486-го компьютера довольно ощутимо

(несколько секунд), хотя абсолютное большинство этих модулей для данного проекта не нужно.

Совет 2.1. Удалите вообще файл AUTOLOAD.MAK (имеется в виду, конечно, — сохраните под другим именем, так как начальный список VBX'ов может пригодиться). Команда New File будет выполняться мгновенно, при этом на экране вы получите минимальный набор элементов управления, который можно будет расширять по мере необходимости с помощью команды Add File".

Совет 2.2. Если вы определились с составом дополнительных элементов управления для своих проектов, то соответственно отредактируйте содержимое файла AUTOLOAD.MAK, удалив из него неиспользуемые строки и добавив свои собственные.

Совет 2.3. Можно сделать несколько собственных начальных заготовок MAK-файлов, создавая новый проект с помощью команды Open Project, а потом сохраняя его под оригинальным именем командой Save As.

Совет 2.4. Если вы долго работаете с одним и тем же проектом, присвойте ему имя AUTOLOAD.MAK, и тогда при запуске VB он будет загружаться сразу.

Совет 3 (глобальный). Используйте функции Windows API

Одним из эффективных средств расширения возможностей Visual Basic является использование многочисленных функций Windows API (Application Program Interface — программный интерфейс приложений Windows). При этом есть два основных аспекта применения Windows API:

- использование новых функциональных возможностей, принципиально не реализуемых на уровне встроенных возможностей языка;
- замена встроенных операторов языка на альтернативные варианты, которые на уровне Windows API зачастую выполняются быстрее.

Наверное, вполне возможно такое сравнение: Windows API для систем программирования в Windows являются примерно тем же, чем для DOSовских систем являются системные функции BIOS/DOS. Отличие заключается в том, что состав функций Windows API гораздо шире (с их помощью, кстати, можно обращаться и к функциям BIOS/DOS) и обращение к ним выполняется с помощью обыкновенных процедурных обращений.

Однако нужно подчеркнуть, что использование функций API имеет и обратную сторону. Как известно, одной из самых сильных черт программирования в среде является надежность создаваемых программ: работая под управлением интерпретатора, они теоретически не могут нарушить работу Windows. Но использование напрямую функций Windows API или других DLLбиблиотек снимает эту защиту (основная проблема — отсутствие контроля за межпроцедурным интерфейсом). Поэтому ошибка в обращении к внешним функциям может привести к неработоспособности системы, что особенно актуально на этапе разработки программы.

Все это требует очень аккуратной работы при использовании функции API. По данному поводу в зарубежной специальной литературе есть целый ряд полезных рекомендаций. Но стоит отметить, что эта проблема аналогична общей постановке вопроса смешанного программирования и использования двоичных библиотек. Данный вопрос, в частности, на примере Basicпрограммирования в среде DOS рассматривался в нескольких номерах журнала "Монитор" — ьь 8'94, 1-2'95.

В состав VB 3.0 входят специальные справочные файлы, посвященные функциям Windows API. Кроме того, файл WIN31API.HLP содержит готовые операторы объявления для функций API, которые можно просто скопировать в текст разрабатываемой программы. В файле WIN31WH.HLP приведено подробное описание всех функций, расположенных в алфавитном порядке имен, поэтому справкой легко пользоваться, если вам известно имя функции. Но если вы хотите посмотреть, есть ли какая-нибудь подходящая функция для решения возникшей проблемы, то сделать это совсем непросто. Аналогичные справочные средства по Windows 95 API имеются в составе VB 4.0.

Следует иметь в виду, что функции API в Windows 95 представляют собой не простое расширение состава аналогичных функций для Windows 3.x. В Windows 95 API изменен интерфейс обращения ко многим "старым" функциям, поэтому в общем случае при переходе от Windows 3.x к Windows 95 потребуется аккуратная коррекция исходного текста программы.

Всем, кто хотел бы более детально познакомиться с проблемой использования функций Windows API в среде Visual Basic, полезно познакомиться с книгой известного американского автора Daniel Appleman "Visual Basic Programmer's Guide for Windows API", которая в течение последних двух-трех лет постоянно входит в число бестселлеров по теме "Visual Basic". В начале книги приводится одно их самых лучших объяснений, что такое Windows и что такое программирование в Windows. Далее дается описание всех 700 функций Windows API, которые разбиты на логические функциональные группы с точки зрения VB-программиста. В начале 1996 года ожидается выход новой книги Appleman на эту же тему для VB 4.0 и Windows 95.

Совет 3а. Как реализовать в VB/Win функции MKx$/CVx

Для многих из тех, кто ранее программировал в DOSовских версиях MS Basic (Quick, PDS, Visual), при переходе в VB/Win неприятным сюрпризом стало отсутствие в нем целой группы очень полезных встроенных функций MKx$/CVx (см. описание MS Basic/DOS). С их помощью производится преобразование числовых (целых, вещественных и пр.) данных в строковый формат, и наоборот. Точнее говоря, никакого преобразования значений здесь нет, а просто N-е количество байт меняет название типа данных.

Основной смысл такого преобразования — возможность хранения и передачи разнотипных данных в виде одной строковой переменной. Это является отличной альтернативой структурам данных с их жестким описанием полей на уровне исходного текста. Обсуждение возможностей использования таких конструкций вполне достойно отдельного разговора, поэтому приведем только один простой пример.

Пример. Необходимо представить адрес смещения в файле, выраженный переменной Offset& (тип LONG), в виде двух беззнаковых двухбайтовых целочисленных переменных. Это часто бывает необходимо для обращения к функциям BIOS/DOS6 и проблема заключается в том, что в MS Basic нельзя использовать беззнаковые переменные в арифметических операциях. Один из простых способов решения этой проблемы выглядит так:

```
' Offset& => HighOffset% & LowOffset%
StringValue$ = MKL$(Offset&)
HighOffset% = CVI(LEFT$(StringValue$, 2))
LowOffset% = CVI(RIGHT$(StringValue$, 2))
'
```

```
' HighOffset% & LowOffset% => Offset%
Offset& = CVL(MKI$(HighOffset%) + MKI$(LowOffset))
```

Однако в VB/Win можно довольно просто реализовать эти полезные Basic-функции с помощью Windows API (хотя лучше, если бы они уже были созданы). Набор таких процедур приведен в модуле MKXCVX.BAS, а пример их применения — в модуле MKXCVXTS.BAS. Они используют очень полезную функцию HMemCpy, которая копирует заданное число байт из одной области памяти в другую. Имеет смысл взять данную функцию на заметку — она может пригодиться и для других операций (например, для быстрого копирования одного массива в другой).

Здесь следует обратить внимание прежде всего на то, что в VB/Win, по сравнению с Basic/DOS, несколько изменилось назначение и формат оператора Declare:

1. В VB/Win этот оператор используется только для обращения к внешним процедурам, представленным в виде DLL-библиотек, и API-функциям (аналог в DOSовских версиях — QLB-библиотеки). В Basic/DOS оператор Declare был нужен и для процедур, загруженных в виде исходных модулей (для процедур-подпрограмм он мог формироваться автоматически, а для функций его надо было писать вручную). Именно поэтому в модуле MKXCVXTS.BAS отсутствуют объявления функций MKx/CVx.
2. В Basic/DOS для одной программы можно было использовать только одну QLB-библиотеку, имя которой указывалось при запуске программы. Для VB/Win-программы может использоваться много различных внешних библиотек. Поэтому в операторе Declare появилось описание имени файла, в котором находится описываемая функция.

Теперь хотелось бы сделать одно замечание. С точки зрения разработчика механизм, реализованный в VB/Win, вроде бы достаточно удобен. Но для работы готовой программы он не оптимален. Представьте себе, если вы используете 10 маленьких процедур, каждая из которых находится в отдельной (и очень большой) DLL-библиотеке. Совершенно очевидно, что это не лучшим образом отразится на быстродействии программы или на требованиях к оперативной памяти. Было бы гораздо удобнее, если бы был реализован традиционный принцип создания исполняемого модуля: выборка на этапе компоновки из двоичных библиотек только нужных процедур и включение их в EXE-модуль или формирование из них оптимальной DLL-библиотеки для данной задачи.

```
'================================================
' Модуль MKXCVX.BAS
'================================================
'
'Реализация функций CVx/MKx$ с помощью обращения к
'функции Windows API HmemCpy (копирование заданного
'числа байт из одной области памяти в другую)
'
'ПРИМЕЧАНИЕ. Передача строковой переменной по значению
'(ByVal) означает, что в функцию HMemCpy передается
'адрес не описателя, а самой строки
'
Declare Sub HMemCpy Lib "kernel" _
  (hpvDest As Any, hpvSource As Any, ByVal cbCopy As Long)
'
Function CVD (x$) As Double
  HMemCpy Temp#, ByVal x$, 8
  CVD = Temp#
End Function

Function CVI (x$) As Integer
```

```
  HMemCpy Temp%, ByVal x$, 2
  CVI = Temp%
End Function

Function CVL (x$) As Long
  HMemCpy Temp&, ByVal x$, 4
  CVL = Temp&
End Function

Function CVS (x$) As Single
  HMemCpy Temp!, ByVal x$, 4
  CVS = Temp!
End Function

Function MKD$ (x#)
  Dim Temp As String * 8
  HMemCpy ByVal Temp, x#, 8
  MKD$ = Temp
End Function

Function MKI$ (x%)
  Dim Temp As String * 2
  HMemCpy ByVal Temp, x%, 2
  MKI$ = Temp
End Function

Function MKL$ (x&)
  Dim Temp As String * 4
  HMemCpy ByVal Temp, x&, 4
  MKL$ = Temp
End Function

Function MKS$ (x!)
  Dim Temp As String * 4
  HMemCpy ByVal Temp, x!, 4
  MKS$ = Temp
End Function

'================================================
' Модуль MKXCVXTS.BAS
'================================================
'
Sub main ()
'
'Пример обращения к функциям MKx$, CVx
'в стиле a la MS Basic for DOS (QB/PDS/VBDOS)
'
  IntValue% = 12345
  x$ = MKI$(IntValue%)
  MsgBox Str$(CVI(x$))
  '
  LongValue& = 12345678
  x$ = MKL$(LongValue&)
  MsgBox Str$(CVL(x$))
  '
  SingleValue! = 123.45
  x$ = MKS$(SingleValue!)
  MsgBox Str$(CVS(x$))
  '
  DoubleValue# = 1.2345
  x$ = MKD$(DoubleValue#)
  MsgBox Str$(CVD(x$))
  '
```

```
End Sub
```

Андрей Колесов

© 1996, Андрей Колесов
Авторский вариант. Статья была опубликована с незначительной литературной правкой в журнале "КомпьютерПресс" N 4/96, с.50-51.

Совет 4. Используйте разнообразные источники информации по Visual Basic

Примечание. *Самую последнюю информацию по этому воросу можно найти по адресу Kolesov's VB & VBA House"*[1].

Отечественные пользователи Visual Basic, скажем прямо, не избалованы вниманием отечественной компьютерной прессы, книжных изданий и самой фирмы Microsoft. Правда, некоторым утешением служит то, что примерно в таком же положении в нашей стране находятся пользователи и других систем программирования.

В 1995 г. в России издано несколько переводных книг (мне лично известны три такие книги) по VB. На мой взгляд, самая удачная из них — "Running Visual Basic 3 for Windows" Р. Нельсона, подготовленная издательством "Русская редакция". Данная книга ориентирована в первую очередь на людей, начинающих освоение VB, тем не менее она может оказаться полезной и для опытных разработчиков. (Употребление термина "начинающие программисты" в этом случае не совсем корректно, так как и для достаточно опытных DOS-программистов переход в Windows- программирование сопряжен с определенными трудностями.) В начале 1996 года вышел повторный тираж этой книги, и сейчас ее можно купить в столичных книжных магазинах. Мне также известно, что две переводные книги по VB версии 4.0 готовятся к изданию в русской редакции PC Week, но выйдут в свет не раньше осени этого года. Вполне вероятно, что появятся и другие книги.

В качестве же радикального решения информационной проблемы я настоятельно советую обратиться к зарубежным источникам. Среди них самым популярным и авторитетным периодическим изданием в области VB-программирования, безусловно, является американский журнал "Visual Basic Programmer's Journal" (VBPJ). Основан он в 1991 г., выпускался один раз в два месяца (до 1993 года — под названием "BasicPro"). С начала 1995 года журнал стал ежемесячным, а его тираж перевалил за 120 тыс. экземпляров. В целом он, конечно, ориентирован на достаточно подготовленных пользователей. Постоянными авторами являются многие технические писатели и разработчики VB-продуктов. Помимо оригинальных материалов, в журнале публикуется масса полезной справочной и рекламной информации. Один из номеров журнала традиционно посвящается ежегодному обзору дополнительных продуктов для VB. С января 1995 года ежеквартально VBPJ выпускает свой CD-ROM.

Одним из наиболее эффективных средств общения программистов являются электронные телеконференции. С конца прошлого года в России работает телеконференция FIDO под

[1] http://www.visual.2000.ru/develop/vb/index.htm

названием RU.VISUAL.BASIC. В настоящее время обсуждается возможность доступа в эту телеконференцию из RELCOM и других российских сетей.

Те, у кого нет проблем с английским языком и выходом в Internet, могут принять участие в международных конференциях сети UseNet на тему Visual Basic. Раньше это была одна конференция — comp.lang.basic.visual, но к конце прошлого года из-за возросшего объема информации она разделилась на четыре:

- comp.lang.basic.visual.announce (официальная информация);
- comp.lang.basic.visual.3rdparty (дополнительные продукты и расширения);
- comp.lang.basic.visual.database (разработка баз данных);
- comp.lang.basic.visual.misc (общие вопросы программирования).

Первая в этой группе конференций, "announce", занимает особое положение. Она доступна пользователям только в режиме чтения, а информация в ней обновляется самим модератором (ведущим конференции). Эта конференция содержит огромный объем самой свежей информации о Visual Basic, в том числе в виде "наиболее часто задаваемых вопросов по VB". Можно гарантировать, что если вы не найдете в ней ответа по своей конкретной проблеме, то, по крайней мере, получите совет, где его можно найти.

Кстати, в UseNet имеются две конференции по другим версиям Basic: comp.lang.basic.misc и alt.lang.basic (девиз этой конференции — ЯЗЫК, КОТОРЫЙ НИКОГДА НЕ УМРЕТ). Огромный объем полезной информации можно получить через информационные ресурсы Internet. (Подробные данные о зарубежных электронных источниках информации можно почерпнуть в конференции comp.lang.basic.visual.announce.)

Разнообразное программное обеспечение PUBLIC DOMAIN, SHAREWARE и FREEWARE вы найдете на FTP-серверах ftp.cica.indiana.edu:/pub/ pc/win3/programr/vbasic (исходные тексты на Visual Basic, утилиты, примеры, VBX, пр.) и ftp.microsoft.com. Последний представляет собой обширный справочный сервер Microsoft общего доступа, в котором наиболее полезными каталогами являются /Softlib/MSLFILES, /MSDN, /MSDN/VBTECH, /DEVTOOLS/LANG/VB/PUBLIC.

Огромное число файлов с "Часто задаваемыми вопросами" (Frequently Asked Questions — FAQ) находятся на общедоступном FTP сервере "rtfm.mit.edu" в каталоге:

ftp://rtfm.mit.edu/pub/usenet/comp.lang.basic.visual.misc/

Альтернативные серверы представлены в следующей таблице:

Местоположение	**Адрес сервера**	**Каталог**
Северная Америка	ftp.uu.net	/usenet/news.answers
Европа	ftp.uni-paderborn.de	/pub/FAQ
	ftp.Germany.EU.net	/pub/newsarchive/news.answers

	grasp1.univ-lyon1.fr	/pub/faq
	ftp.win.tue.nl	/pub/usenet/news.answers
	ftp.sunet.se	/pub/usenet
Азия	nctuccca.edu.tw	/USENET/FAQ
	hwarang.postech.ac.kr	/pub/usenet/news.answers

Желающие могут автоматически получать текст периодически обновляемого FAQ по электронной почте. Для этого необходимо отправить сообщение на почтовый сервер "mailserver@rtfm.mit.edu" следующего содержания:

send usenet/comp.lang.basic.visual.misc/*

Много полезной информации содержат электронные форумы сети CompuServe, в частности MS BASIC Forum (Microsoft) и VBPJFO Forum (журнала VBPJ). Кроме того, пользователи VB имеют возможность получать из сети Internet исправленные варианты файлов VB. В последнее время наиболее активно развивающимся источником информации является World Wide Web, где, в частности, почти все фирмы-разработчики и продавцы программных продуктов имеют свои домашние страницы. Самые популярные WWW-серверы:

```
http://www.apexsc.com/vb/vb4.html - Carl & Gary's page
http://www.microsoft.com/kb/faq/  - Microsoft's FAQ
http://www.windx.com              - Visual Basic Programmer's Journal
http://www.iadfw.net/gbeene/      -  Gary Beene's page
```

В США постоянно имеются в продаже три-четыре десятка разнообразных книг по VB. В связи с выходом VB 4.0 осенью прошлого года появилось почти два десятка новых изданий. Как правило, все книги имеют объем 800-1200 страниц и обязательно включают программные приложения на дискете или компакт-диске. Стоимость каждой книги — 35-45 долл.

Довольно полный перечень книг можно найти в каждом номере VBPJ, а также в телеконференциях, в первую очередь в comp.lang.basic.visual.announce. Особую ценность информации в этих конференциях придают публикуемые в них отзывы самих программистов.

Во всех перечисленных выше источниках содержится большой объем информации о новой версии VB 4.0. В первую очередь рекомендую познакомиться с октябрьским и ноябрьским номерами журнала VBPJ за прошлый год, которые посвящены соответственно обзорам VB 4.0 в целом и редакции Enterprise Edition.

Совет 5. Выполните "очистку" своего проекта перед созданием окончательного варианта EXE-модуля

При создании EXE-модуля, после работы с проектом в среде VB, в него попадает некоторый дополнительный "мусор", так как VB не полностью очищает структуры и переменные, которые использовались в процессе разработки, хотя после этого и были удалены. Для такой "очистки" проекта сохраните его, а потом вновь запустите VB (еще лучше — и Windows), чтобы в него попали только реально используемые модули приложения. Вполне возможно, что эта простая операция уменьшит размер исполняемого модуля на 10-30%.

Совет 6. Будьте внимательны при использовании оператора DIM

Дело в том, что в Basic используется синтаксис описания переменных, несколько отличный от принятого, например, в Pascal. Поэтому программисты, имеющие опыт работы на Pascal, могут попасть впросак, написав:

```
Dim iA, iB, iC as Integer
```

При этом они думают, что все три переменные будут иметь тип Integer. В действительности для первых двух будет установлен тип данных по умолчанию, обычно Variant. Вместо этого необходимо использовать один из следующих двух способов:

```
Dim iA as Integer, iB as Integer, iC as Integer
```

или

```
Dim iA%, iB%, iC%
```

Совет 7. Сохраняйте свои программы в формате TEXT, а не BINARY

Сохранение в двоичном формате происходит намного быстрее, но почему же рекомендуется это делать в текстовом формате? Есть несколько доводов в пользу последнего варианта.

- Если исходный код и проект сохраняются как текст, то он превращается в ASCII-код, который вы можете редактировать в любом текстовом процессоре. А двоичный код может читаться только в среде разработки VB (текущей или последующих версий). Например, если вы используете ASCII-код, то даже с помощью самого простого текстового редактора можно вырезать и вставлять код из других исходных модулей или модулей форм.
- Если вы захотите распечатать свой проект на бумаге, то использование команды Print из меню File в среде разработки VB не является оптимальным вариантом. Для этого лучше импортировать текстовые файлы в наиболее удобный для вас текстовый процессор.
- Программа Setup Wizard не может сканировать проекты, записанные в двоичном формате.
- И наконец, если что-то не так (изменился только один байт!), то это вызовет серьезные проблемы при работе в двоичном режиме. В текстовом режиме вам будет намного проще исправить такую ошибку.

По опыту работы с версиями DOS Basic известно, что компилятор (при создании EXE-модуля) периодически "спотыкался" на некоторых конструкциях программы, записанных в формате BINARY, выдавая довольно непонятную диагностику. Это в какой-то степени объяснимо: компилятор и интерпретатор — это две разные

программы, каждая из которых имеет свои собственные ошибки. Наверное, именно поэтому разработчики Microsoft в версии Basic PDS выбрали формат TEXT в качестве используемого по умолчанию формата сохранения исходного текста.

Андрей Колесов

© 1996, Андрей Колесов
Авторский вариант. Статья была опубликована с незначительной литературной правкой в журнале "КомпьютерПресс" N 5/96, с. 80-83.

Совет 8. Читайте литературу, русскую и американскую

При освоении системы программирования, тем более новой, нужна хорошая литература. Безусловно, помогает документация и встроенная справочная система, но этого явно недостаточно для нормальной работы. Во-первых, я абсолютно уверен, что у большинства пользователей VB в России документации просто нет. (Рекламная пауза: "Покупайте лицензионные продукты.") Во- вторых, назвать ее исчерпывающей и до конца понятной — никак нельзя. Здесь лучше всего вспомнить о том, что Коран — одна книга, а толкования к нему насчитывают тысячи томов. И в- третьих, оказывается, что довольно большое число отечественных программистов предпочитают описания на русском языке.

В начале марта в продаже появилась новая книга на тему VB: "Visual Basic 3.0, Visual Basic 4.0 для Windows" (Москва, ABF, 1996, 360 стр., тираж 11 тыс. экз.) Имя ее автора, Хачатура Арушанова, хорошо известно в среде российских VB-программистов по публикациям в прессе и активной деятельности в сети FIDO. Книга предназначена в первую очередь для начинающих VB- программистов. Основная ее часть — систематический справочник по языку (описание операторов), и в этом плане она очень хорошо дополняет книгу "Running Visual Basic 3 for Windows", о которой я упоминал в Совете 4. Вообще я должен сказать, что написание книги о VB является делом непростым. Здесь важно определить точку отсчета — базовые знания языка читателя. Для меня, как человека, имеющего опыт программирования в QuickBasic, операторы VB процентов на 80 давно известны, и мне интереснее прочитать о свойствах, методах и пр. Однако многие VB-программисты незнакомы с базовым языком.

Книга Арушанова, кроме справочника, содержит и другие интересные сведения, советы и пр. В общем — книга полезная, и здесь мое мнение совпадает с мнением многих моих знакомых, прочитавших ее. Но все же надо сказать, что вторая часть заголовка "... Visual Basic 4.0" не должна вводить в заблуждение читателя — в статьях самого Арушанова, опубликованных ранее в "ComputerWeek", говорится о VB 4.0 гораздо подробнее. Впрочем, это не должно расцениваться как упрек автору: VB 4.0 — это целый мир, охватить который в одной книге просто невозможно.

В связи с этим следует упомянуть, что в США в продаже постоянно имеется несколько десятков книг, освещающих самые различные аспекты программирования на VB (базы данных, мультимедиа, классы, объекты и пр). Как правило, они имеют объем от 500 до 1200 стр., включают дискету или CD-ROM с программными примерами и стоят от 30 до 45 долл. В каждом номере американского журнала VBPJ в специальном разделе публикуются аннотации этих книг. В конце прошлого года, в связи с выходом VB 4.0,

появилось огромное количество книг, посвященных этой версии. Среди них есть и переиздания, и принципиально новые книги, число которых продолжает расти. Я привожу здесь информацию об американских изданиях не с целью подразнить российских пользователей "их" жизнью, а для того, чтобы сказать, что при желании все эти книги можно реально приобрести. Готов поделиться собственным опытом.

Совет 9. Используйте оператор Option Explicit

Наличие оператора Option Explicit (он находится в части Declarations программного модуля) свидетельствует о том, что все переменные данного модуля должны быть объявлены с помощью операторов DIM или REDIM. При появлении необъявленных переменных выдается сообщение об ошибке при запуске программы.

Для управления автоматической установкой этого режима во всех создаваемых модулях и формах установите флажок Reqiure Variable Declaration в положение Yes/No. Это флажок находится в диалоговом окне Environment Options, обращение к которому выполняется через соответствующую команду в меню Tools (VB 4.0) или Options (VB 3.0).

Отношение программистов к необходимости этого оператора непосредственно зависит от их квалификации. В частности, большинство профессионалов считают, что такой функции вообще не должно быть, так как явное описание переменных должно быть обязательным всегда. По их мнению, отсутствие ранее подобного режима — один из явных признаков "примитивности" Basic.

Традиционно Basic (и не только он, но и, например, Fortran) использовал принцип "по умолчанию". Его смысл заключался в том, что первое появление названия переменной в тексте программы автоматически означало ее объявление — самого существования переменной и ее типа. Вроде бы, это облегчало жизнь программистам (не надо было писать одну-две строки в заголовке модуля), но проблемы, возникающие при отладке из-за неправильного написания имен переменных, могли быть весьма значительными. Приведем примеры таких ситуаций.

```
i% = 1: m = I
```

Здесь программист забыл поставить в идентификаторе знак "%", в результате появилась новая переменная i, а значение m стало равным 0, а не 1.

```
MainCount = 1: m = MaimCount
```

То же самое — ошибка в написании идентификатора из-за довольно сложного имени (бывает и сложнее).

Обнаружить такие ошибки в больших программах бывает непросто. Этих проблем не возникло бы, если бы был установлен режим Option Explicit и эти переменные были бы объявлены:

```
Dim i%, MainCount
```

Явное описание переменных является хорошим поводом подумать об оптимальном использовании переменных в программе. Еще один пример:

```
For i = 1 To 10
```

```
   ...
   If A > B Then i = 1 Else i= 2
Next
```

В данном случае программист забыл, что переменная i является счетчиком цикла, и задействовал ее в качестве флажка. В классическом варианте описание переменной должно было выглядеть так:

```
Dim i ' i - счетчик цикла
```

Такое описание подсказало бы программисту, что переменная i уже использована в других целях.

Совет 10. Следите за правильным использованием типов данных

Смысл этого простого совета заключается в следующем. Время выполнения операций для числовых данных разных типов различается довольно значительно. Кроме того, для их хранения требуются различные размеры памяти. Поэтому следует выбирать оптимальный тип переменной в зависимости от ее применения. Это особенно актуально для больших программ, но привыкать к этому нужно сразу. Не забывайте, например, что переменные Integer всегда работают значительно быстрее, чем Single и Variant.

В связи с этим следует напомнить, что в VB имеются три варианта определения типа переменной:

```
Dim Variable[суффикс - %,#,!,&,$,@]
Dim Variable As ....
Dim Variable
```

1. Объявление типа с помощью суффикса является "исторической" спецификой Basic, лет десять назад других вариантов просто не было. Вообще-то, это было довольно удобно — по имени переменной сразу был виден ее тип. До появления VB в литературе по Basic этот классический вариант всегда использовался в листингах программ. Но расширение числа типов данных (в VB 4.0 их уже двенадцать) сделало невозможным дальнейшее применение данного принципа объявления типа переменной.
2. Вариант "As" — наиболее универсальный, именно он считается сегодня методически самым верным.
3. К третьему варианту нужно относиться с осторожностью. В этом случае тип переменных зависит от имени первой буквы. В VB имеется возможность делать такие установки с помощью группы операторов типа DefXXX (Defint, Defsng и пр.). Следует помнить, что по умолчанию, начиная с VB 3.0, тип переменной является Variant (в более ранних версиях — Single).

Современный стиль программирования предполагает использование смысловых идентификаторов (например, "FileNumber", а не "Ifn"), поэтому такой способ определения типа переменной применяется все реже. Тем не менее он очень удобен для новой установки по умолчанию всех переменных модуля. Чаще всего это делается для установки целочисленного типа: Defint I-Z.

Совет 11. Используйте оптимальные программные конструкции

Собственно это продолжение предыдущего совета.

Сначала некоторые общие соображения. Учитывая, что создание программы должно укладываться в определенные сроки, общая стратегия разработки строится следующим образом. Прежде всего как можно быстрее создается первый работающий вариант, а затем выполняется доводка, в первую очередь фрагментов программы, наиболее критичных с точки зрения быстродействия. Здесь достаточно четко действует известный закон статистики 20/80 ("20% населения выпивают 80% пива") — "20% программы занимают 80% времени ее выполнения" и соответственно "20% программы требуют 80% времени на их разработку". При этом, разумеется, не стоит пренебрегать "правилами хорошего тона" и в процессе написания первого варианта программы. В связи с этим очевидный совет: будьте особенно внимательны при написании больших циклов.

Рекомендации, касающиеся кода, связанного с обработкой числовых данных, хорошо известны:

- без нужды не используйте вещественные числа, а тем более с двойной точностью (целочисленная арифметика работает на порядок быстрее);
- избегайте смешения типов переменных (целых и вещественных) в арифметических выражениях (много времени тратится на преобразование форматов);
- помните, что время выполнения арифметических операций возрастает в таком порядке: сложение, умножение, деление, возведение в степень (выражения A+A, A*.5 и A*A будут вычисляться быстрее, чем соответственно 2*A, A/2 и A^2).

Что касается строковых переменных, что специфика работы с ними заключается в том, что они являются динамическими, поэтому самая длинная операция с ними — это простое присвоение: A$=... В этом случае каждый раз идет обращение к внутреннему диспетчеру динамической памяти, который фактически формирует новую переменную. Поэтому основной совет при работе со строками заключается в следующем: избегайте операций присвоения.

Как это сделать?

Довольно часто встречающейся программной конструкцией является формирование строковой переменной известной длины. Например, нужно сформировать строку, каждый байт которой равен его номеру в строке. Здесь можно использовать такие решения:

Вариант 1:

```
a$ = "": For n% = 1 To 255: a$ = a$ + Chr$(n%): Next
```

Вариант 2:

```
a$ = Space$(255): For n% = 1 To 255: Mid$(a$,n%) = Chr$(n%): Next
```

Второй вариант подлиннее, но будет работать существенно быстрее. Такой способ можно использовать и для случая, когда длина строки до начала цикла ее формирования неизвестна — сначала резервируется строка заведомо большей длины, а после окончания цикла выделяется нужное количество символов оператором LEFT$.

Совет 12. Пишите комментарии в программе

О пользе комментариев можно говорить много. В данном случае просто имейте в виду: комментарии не включается в состав исполняемого модуля и не влияют на его размеры и скорость выполнения.

Совет 13. Копируйте структуру меню в разные формы

Существует простой способ копирования сходных структур меню в несколько разных форм. Если сохранить файл как текст, тогда его секцию меню можно будет легко вставить в секцию меню нового файла с помощью любого текстового редактора.

Совет 13a. Как выделить текст при попадании в поле

Во многих случаях бывает необходимо выделить или подсветить целиком весь текст поля, когда на него устанавливается фокус. Тогда пользователь может просто начать ввод текста, чтобы заменить исходный текст на новый, или нажать клавишу Tab, чтобы оставить его как есть и перейти к следующему полю. Для выполнения этого требуется написать совсем немного кода. В общем модуле напишите следующий код:

```
Sub SetSelected()
  Screen.ActiveControl.SelStart = 0
  Screen.ActiveControl.SelLength = _
    Len(Screen.ActiveControl.Text)
End Sub
```

А для каждого текстового поля включите такой текст:

```
Sub txtField_GotFocus()
  SetSelected
End Sub
```

Совет 13b. Как реализовать режим замены в текстовом окне

Текстовые окна в Windows всегда работают в режиме вставки и не обеспечивают возможность работы в режиме замены. Однако последний можно легко эмулировать, как показано здесь:

```
Sub Text1_KeyPress(KeyAscii As Integer)
   If KeyAscii >= 32 Then
      If Text1.SelLength = 0 Then
         If Text1.SelStart < Len(Text1) Then
            Text1.SelLength = 1
         End If
      End If
   End If
End Sub
```

Если вводится какой-либо символ (при условии, что текст не был выделен), данный код выбирает текущий символ, устанавливая свойство SelLength равным 1, и позволяет тем самым заменить его вводимым символом.

Совет 13c. Как сделать, чтобы клавиша Tab обрабатывалась как обычный символ

Для этого нужно установить свойство TabStop = False для всех элементов управления в активной форме. После этого вы сможете вводить символы "tab" (chr 9) в текстовые поля элементов управления. Если нужно, чтобы клавиша Tab "вела себя обычным образом" (то есть осуществляла переход к следующему элементу управления) для одного из элементов управления такой формы, то достаточно просто эмулировать ее работу с помощью следующего кода:

```
Sub Command2_KeyDown (KeyCode As Integer, _
  Shift As Integer)
  ' На примере 3-х командных кнопок
  If KeyCode = 9 Then
    If Shift = 0 Then
      Command1.SetFocus
      ' Tab - следующий элемент управления
    ElseIf Shift = 1 Then
      Command3.SetFocus
      ' Shift+Tab - предыдущий
    End If
  End If
End Sub
```

Андрей Колесов, Ольга Павлова

© 1996, Андрей Колесов, Ольга Павлова
Авторский вариант. Статья была опубликована с незначительной литературной правкой в журнале "КомпьютерПресс" N 6/96, с. 58-62.

В настоящее время (весна 1996 года) программисты используют две версии VB — 3.0 и 4.0. Соответственно все программные конструкции, которые предлагаются вниманию читателей, можно разделить на три категории: работающие только в VB 3.0 (таких очень немного), только в VB 4.0 (здесь еще есть такие, которые работают только в Windows 95) и в обеих версиях. Хотя в будущем в основном будут рассматриваться советы, связанные со спецификой VB 4.0, пока договоримся придерживаться такого правила: по умолчанию примем, что совет относится к обеим версиям, в противном случае в заголовке совета появится соответствующее указание.

Следующий выпуск будет ориентирован в основном на VB 4.0. Там мы, в частности, предполагаем рассмотреть некоторые вопросы создания объектов и OLE-серверов.

Совет 14. Выделение нескольких имен файлов в диалоговом окне Open File

Если вы хотите, чтобы в диалоговом окне выводилось несколько имен файлов, установите флажки OFN_ALLOWMULTISELECT. В результате этого строка FileName будет состоять из имени диска и имени каталога, за которыми последует список файлов, выбранных в данном каталоге и разделенных между собой пробелами. Например, если Вы выберете файлы X.TXT и Y.TXT, находящиеся в каталоге c:\a\b, то получите имя файла c:\a\b\ x.txt y.txt, а НЕ c:\a\b\x.txt c:\a\b\y.txt.

Совет 15. Закрывайте другую программу из своего приложения

Для закрытия другой программы используйте следующий программный код в тексте своего приложения:

```
Title = "Программа А"
ihWnd = FindWindow(0&, Title)
ihTask = GetWindowTask(ihWnd)
iRet = PostAppMessage(ihTask, WM_QUIT, 0, 0&)
MsgBox "Внимание! Программа А закрывается"
```

Совет 16. Удаляйте код после удаления соответствующего элемента управления

При удалении какого-либо элемента управления из формы код, соответствующий этому элементу управления, не удаляется. (Об этой проблеме мы уже говорили раньше в связи с переименованием элементов управления.) Он по-прежнему хранится в программном коде в Общей области (General area), и хотя он никому не мешает, из-за него увеличивается размер исполняемого модуля. Кроме того, если вы создадите новый элемент управления с таким же именем, сохранившийся код будет принадлежать этому новому компоненту, так как событие имеет то же самое имя. Поэтому программист, который не хочет сохранять данный код (а кто хотел бы?), должен удалять его вручную каждый раз при удалении элемента управления. Непонятно только одно: почему Microsoft не предлагает автоматическое удаление подобного мусора?

Совет 17. Производите очистку данных при выгрузке форм

Для выгрузки формы можно использовать следующий фрагмент кода, закрепленный за кнопкой Close:

```
Unload
```

А этот код следует также поместить в обработчик события Form_Unload():

```
Set  = Nothing
```

Такая операция очищает сегмент данных выгружаемой формы от любого оставшегося кода или данных, если вы не делаете этого в явном виде. Это особенно важно при использовании большого числа форм, тем более если на них расположено много элементов управления.

Совет 18. Реагируйте на появление ошибок GPF правильно

Появление сообщений об ошибке GPF (General Protection Fault) — вещь неприятная. К счастью, в VB они появляются существенно реже, чем в компилирующих системах программирования. По теории они могут появляться в VB только при неправильных обращениях к внешним динамическим библиотекам и функциям API.

Сообщения об ошибках могут иметь следующий вид:

- Application error: VB caused a General Protection Fault in VB.EXE at nnnn:nnnn
- Assertion failed

- Bad handle
- Bad heap block

Это может произойти, если выполняется любое из следующих условий:

- неправильное расположение слова ByVal в операторе Declare;
- длина передаваемой строки, получаемая при инициализации, слишком коротка для получения возвращаемого значения;
- неопределенные параметры в объявлении или вызове функции;
- неправильный тип или длина параметров в объявлении или вызове функции.

Если вы получаете сообщения об ошибках GPF при обращении к DLL и не можете сразу установить причину, тогда напишите программу-заглушку в виде BAS-файла, которая имитирует DLL-библиотеку. Это поможет хотя бы решить, кто виноват — библиотека или ваша VB- программа. Если ошибка все-таки остается, причиной ее является не DLL-библиотека.

Совет 19. Изменить последовательность прохождения элементов управления совсем не трудно

При разработке программы последовательность прохождения элементов управления может нарушиться, особенно если в форму добавляется какой-либо элемент управления после того, как все остальные уже находятся на своих местах. В этом случае вам потребуется установить новую последовательность TabOrder. Для этого при помощи мыши выделите по очереди все элементы управления, двигаясь в обратном порядке, и для каждого из них установите значение свойства TabOrder равным нулю. В результате последовательность элементов управления станет обратной той, в которой вы их проходили: теперь они будут следовать в необходимом порядке.

Совет 20. Избегайте длинных имен элементов управления и форм

Помните, что имена элементов управления и форм хранятся в исполняемом файле, зато на именах переменных можете не экономить: они не хранятся в EXE-файлах.

Совет 21. Используйте Esc для выхода из программы-справки

При создании файлов справки для VB-программ полезно предусмотреть возможность того, чтобы пользователь мог легко выйти из окна справки и вернуться обратно в VB-приложение. Один из способов реализации такой возможности — при помощи клавиши Esc завершить выполнение программы-справки. Для этого добавьте следующий код в секцию [CONFIG] в файле проекта справки (help project file). Данный код создает клавишу-акселератор (Esc), чтобы активизировать в справочной системе макрокоманду EXIT, которая завершает выполнение программы-справки.

```
[CONFIG]
; выход из программы-справки по клавише Esc
AddAccelerator(0x1B,0,"Exit()")
```

Хорошо было бы прочитать этот совет самим разработчикам среды VB.

Совет 22. Иногда полезно редактировать MAK-файл вручную

VB не предоставляет пользователю простого способа редактировать содержимое MAK-файлов. Но это может понадобиться, если вы хотите добавить новый VBX в середину разрабатываемого проекта или быстро убрать ненужные модули из проекта. Особенно это бывает полезно, когда вы пользуетесь компьютером, на котором VBX'ы, необходимые для работы вашего приложения, установлены в других каталогах. В данном случае используйте любой текстовый редактор для ручной настройки MAK-файла.

Совет 23. Разбивайте большие приложения на части

Если вы создали большое приложение, подумайте о том, как разбить его на более мелкие. Используйте только одну главную исполняемую программу, которая вызывает другие программы, а также DDE-технологию для осуществления связей. Когда вы открываете неглавное приложение, откройте DDE-связь с главным приложением. Вы можете написать сценарий, который позволит приложениям взаимодействовать друг с другом для передачи данных или начала выполнения процессов. Такой метод имеет несколько преимуществ, включая более низкие требования к размеру памяти и меньшее потребление ресурсов. И еще одно БОЛЬШОЕ преимущество: поскольку ваше приложение состоит из нескольких исполняемых модулей, Windows выделяет для выполнения каждого из них свой собственный интервал времени, за счет чего это приложение выполняется быстрее.

Совет 24. Используйте префикс в именах переменных

В Совете 10[2] мы уже говорили, что раньше для обозначения типа переменных и функций в Basic можно было использовать суффикс. Это было очень удобно: по имени сразу было видно, с какой переменной имеешь дело. Но с расширением числа типов в VB специальных символов (%, #, $, и пр.) для суффиксов просто не стало хватать, и поэтому такой способ обозначения переменных в последнее время не используется.

Идея быстрого опознавания переменных довольно просто реализуется с помощью префиксов (приставок в начале названия переменной). Здесь нет никаких строго обязательных правил, и каждый может использовать собственную систему обозначений. Но лучше все-таки применять общепринятые обозначения — тогда вы будете лучше понимать чужие тексты, а ваши коллеги — ваши программы. О реализации этой идеи применительно к названиям элементов управления мы уже писали в самом первом своем Совете[3].

В качестве рекомендуемых префиксов предлагаются такие сокращения:

- Типы данных и функций (три символа: именно эти сокращения используются в модификациях оператора DefXXX — определение типа переменной по первой букве): Byte — byt, Boolean — bln, Collection — clt, Currency — cur, Date (Time) — dat (tim), Double — dbl, Error — err, Integer — int, Long — lng, Object — obj, Single — sng, String — str, User-defined — udt, Variant — var.
- Объекты базы данных (две буквы): DataBase — db, Dynaset — ds, Field — fd, Index — ix, QueryDef — qd, Query — Qu, Table — tb, TableDef — td.

[2] http://www.visual.2000.ru/develop/ms-vb/tips/9605.htm
[3] http://www.visual.2000.ru/develop/ms-vb/tips/9603.htm

Сейчас общепринятым считается такое правило написания переменных: строчными буквами пишется префикс (это сразу отличит переменную от зарезервированного слова языка, которое автоматически начинается с прописной буквы), после которого с большой буквы идет уникальное имя переменной (лучше, чтобы имя передавало ее смысл):

```
intTimes, strFileName, varSum
```

Там, где есть однозначность, вполне допускается использование первой буквы префикса (только для переменных, чтобы не было путаницы с объектами баз данных): iTimes, vSum.

В этом случае можно использовать для автоматического обозначения типов оператор DefXXX. Тогда вместо длинного описания:

```
Dim Times As Integer, Sum As Integer, Counter As Integer
```

можно будет написать:

```
DefInt I Dim iTimes, iSum, iCounter
```

Мы думаем, что те, кто привык к суффиксам, вполне может использовать и их — текст программы получается достаточно понятным:

```
Sum = 0: Times% = 10: FileName$ = "program.bas"
```

Андрей Колесов, Ольга Павлова

© 1996, Андрей Колесов, Ольга Павлова
Авторский вариант. Статья была опубликована с незначительной литературной правкой в журнале "КомпьютерПресс" N 7/96, с. 46-50.

Как мы и обещали в прошлый раз, все приведенные в этом выпуске советы предназначены для VB 4.0.

Совет 25. Используйте программный код для прекращения работы Windows 95 и даже для перезапуска Windows

В Windows 95 для этого существует функция ExitWindowsEx, входящая в состав Win 32 API. Данная функция позволяет перезапустить компьютер, прекратить работу, принудительно выйти из системы или осуществить выход, а затем вход другого пользователя.

Первый параметр этой функции представляет собой константу, входящую в список допустимых значений, которая указывает, какую операцию нужно выполнять. Второй параметр является зарезервированным значением и всегда должен быть равен нулю.

Следующий код демонстрирует, как вызвать функцию ExitWindowsEx для перезапуска компьютера:

```
Option Explicit
Private Declare Function ExitWindowsEx Lib "user32" (ByVal _
  uFlags As Long, _ ByVal dwReserved As Long ) As Long
```

```
Private Const EWX_FORCE=4 ' Принудительный выход
Private Const EWX_LOGOFF=0 ' Выход из системы
Private Const EWX_REBOOT=2 ' Перезапуск компьютера
Private Const EWX_SHUTDOWN=1 ' Прекращение работы

Sub Command1_Click ()
  Dim nRet As Long
  '- Перезапуск компьютера
  nRet=ExitWindowsEx (EWX_REBOOT, 0&)
  If nRet=False Then
    MsgBox "Невозможно выйти из Windows"
  End If
End Sub
```

Совет 26. Используйте клавиши управления курсором для настройки расположения или размеров элементов управления

Эта возможность VB 4.0 почти не отражена в документации. Чтобы передвинуть элемент управления, выделите его, а затем, удерживая клавишу Ctrl в нажатом состоянии, работайте с клавишами управления курсором. Аналогично, с помощью клавиши Shift можно изменить размеры элемента управления. Используя этот прием, нетрудно передвинуть или изменить размеры целой группы элементов управления. Если пометить параметр Align Controls to Grid в диалоговом окне Environment Options (это окно появляется, если в меню Tools выбрать команду Environment Options...), тогда каждое нажатие клавиши управления курсором будет приводить к увеличению позиции/размера на один шаг сетки. Если же этот параметр не помечен, элементы управления будут двигаться/менять свои размеры на один пиксел при каждом нажатии клавиши управления курсором.

Совет 27. Создавайте дополнительные средства (add-ins)

Чтобы получить дополнительное средство, необходимо сначала создать VB-приложение в виде OLE-сервера. Для этого нужно использовать соответствующие модули класса для обработки свойств и методов.

Чтобы Visual Basic знал о возможности использования созданного дополнительного средства, оно должно быть зарегистрировано в реестре, для чего в соответствующую секцию файла VB.INI нужно вставить идентификатор ProgID. ProgID состоит из имени проекта, точки и имени класса. Если проект носит имя Junk, а класс - Designer, то идентификатор ProgID будет Junk.Designer. Этот идентификатор следует добавить к секции [Add-Ins16] или [Add-Ins32] в файле VB.INI, в результате он будет выглядеть примерно так:

```
[Add-Ins16]
Microsoft.DesignSpec=0
Junk.Designer=0
Spacer.Connection=1

[Add-Ins32]
Junk.Designer=0
Spacer.Connection=1
```

С помощью значений 0 или 1 в строках INI-файла определяется, следует ли активизировать данное дополнительное средство при работе VB. В случае, если оно было

создано при помощи самого VB, последний вызывает метод ConnectAddIn для данного класса и передает указатель объекту Application в этом методе. Это придает дополнительному средству статус базового, или корневого, объекта, необходимый для работы с объектами более низкого уровня.

Совет 28. Как определить, закончено ли выполнение 32-разрядной программы, запущенной из вашего приложения

Для этого теперь не нужно вызывать функцию API GetMOduleUsage. Более того, лучше использовать функцию CreateProcess() вместо функции Shell() для запуска программы. Функция CreateProcess() обеспечивает возможность управления запускаемой программой посредством одного из передаваемых параметров. Сразу после запуска процесса с помощью обращения к функции CreateProcess() передайте управление к функции WaitForSingleObject(). В результате ваше приложение будет приостановлено до тех пор, пока не прекратит работу запущенный им процесс.

Совет 29. Как работать с наборами элементов (collections)

Microsoft включила в версию 4.0 также несколько новых языковых конструкций для наборов элементов. С помощью нового оператора With ... End With можно укоротить ссылку к полному имени объекта для каждого свойства и метода. Следующий код использует полное имя объекта для каждого имени и свойства:

```
tipNew.ID="T: & Format$(iID+1)
tipNew.Text="Новое свойство может оказаться забавным!"
m_colTips.Add tipNew, tipNew.ID
```

Вы можете укоротить ссылки на объект и ускорить их выполнение с помощью оператора With:

```
With tipNew
  .ID="T: & Format$(iID+1)
  .Description="Новое свойство может оказаться забавным!"
  m_colTips.Add tipNew, .ID
End With
```

Другая новая языковая конструкция - это оператор For Each ... Next. Он аналогичен оператору For ... Next, но отличается тем, что особым образом выбирает каждый элемент, входящий в набор:

```
Private m_colTips as New Collection
Private m_Tip as CTip

For Each m_Tip in m_colTips
   m_Tip.Reserve
Next
```

Совет 30. Оптимизируйте процедуры установки значений свойств

При создании удаленных объектов OLE Automation особое внимание следует уделить вопросу быстродействия. Каждый раз, когда приложение-клиент устанавливает или получает значение свойства, производится вызов механизмов OLE. Поэтому, чтобы увеличить быстродействие сервера OLE Automation, следует уменьшить количество вызовов со стороны приложения- клиента. Например, довольно часто в приложениях-

клиентах бывает необходимо устанавливать и осуществлять поиск нескольких свойств. В этом случае полезно создать специальный метод, который может иметь дело с значениями нескольких свойств - устанавливать их и осуществлять их поиск.

Создание метода PropertyGet позволило бы приложению-клиенту осуществлять поиск текущих значений всех свойств конкретного OLE-сервера. Этот метод содержит список необязательных параметров, в который заносятся все значения свойств, выбранные OLE-сервером для того, чтобы сделать их доступными приложению-клиенту. Поскольку каждый запрос к удаленному OLE- серверу должен пройти по сети, данный метод увеличивает производительность, так как исключаются индивидуальные запросы значений каждого свойства, которые тормозят передачу данных в сети.

Для более полного решения проблемы следует также создать метод PropertySet, который позволил бы приложению-клиенту обновлять значения всех свойств OLE-сервера. Эти процедуры используют функцию IsMissing, чтобы определить, какой из параметров, переданных приложением-клиентом, получает или возвращает текущее значение свойства связанного объекта.

В качестве примера рассмотрим OrderMgr - объект OLE Automation, который может использоваться как часть приложения обработки очереди. Он обращается к объекту Order, который в свою очередь подвергает воздействию пять свойств OrderID, OrderDate, CustomerID, ShippedFlag и RemovedFlag. Объект Order будет работать с этими пятью отдельными свойствами, а также с методами PropertySet и PropertyGet, которые позволяют клиенту более эффективно работать со значениями свойств. Используя их, можно обновлять или осуществлять поиск всех пяти свойств объекта Order с помощью только одного вызова процедуры. Без этой процедуры потребовалось бы пять отдельных вызовов, чтобы получить всю необходимую информацию. Следующий листинг показывает, как в этом примере будут выполняться методы PropertySet и PropertyGet:

```
Public Sub PropertyGet(Optional OrderID As Variant, _
  Optional OrderDate As Variant, _
  Optional CustomerID As Variant, _
  Optional ShippedFlag As Variant, _
  Optional RemovedFlag As Variant)

  If IsMissing(OrderID) = False Then
     OrderID = prvOrderID
  End If
  If IsMissing(OrderDate) = False Then
     OrderDate = prvOrderDate
  End If
  If IsMissing(CustomerID) = False Then
     CustomerID = prvCustomerID
  End If
  If IsMissing(ShippedFlag) = False Then
     ShippedFlag = prvShippedFlag
  End If
  If IsMissing(RemovedFlag) = False Then
     RemovedFlag = prvRemovedFlag
  End If
End Sub

Public Sub PropertySet(Optional OrderDate As Variant, _
  Optional CustomerID As Variant, _
  Optional RemovedFlag As Variant)

  If IsMissing(OrderDate) = False Then
```

```
    prvOrderDate = OrderDate
  End If
  If IsMissing(CustomerID) = False Then
    prvCustomerID = CustomerID
  End If
  If IsMissing(RemovedFlag) = False Then
    prvRemovedFlag = RemovedFlag
  End If
End Sub
```

Совет 31. Следите за тем, что вы передаете в OLE-сервера

Не используйте в библиотеке объектов Visual Basic сами объекты в качестве параметров или возвращаемых значений для подвергаемых воздействию свойств или методов в классах типа Public. Эти объекты предназначены для применения только внутри единичного проекта. Иное их использование может привести к непредсказуемым результатам.

Совет 32. Снабжайте документами свои OLE-серверы

Снабдите документами интерфейс к своему OLE-серверу, используя кнопку Options в Object Browser. Чтобы документация была более полной, используйте для этого специальный файл справки. С помощью Object Browser подсоедините этот файл к OLE-серверу.

Совет 33. Что делать с ошибками OLE-сервера

Не выводите на экран сообщение об ошибках из самого OLE-сервера. Вместо этого передайте его в клиентское приложение. Для нумерации ошибок используйте значения больше чем vbObjectError + 512, и меньше чем vbObjectError + 5535. Значения, которые находятся в диапазоне от vbObjectError до vbObjectError + 512, могут вступить в противоречие со значениями ошибок OLE. Обязательно внесите номера ошибок в файл справки, который вы используете для своего OLE- сервера.

Совет 34. Как обрабатывать объекты OLE-сервера

Не возвращайте ссылки на объекты Form и Control из своего OLE-сервера. Лучше используйте упаковщики (wrappers) для свойств и методов, принадлежащих формам и элементам управления, которые должны обрабатываться вашим OLE-сервером. Например, вместо того чтобы возвращаться в текстовое окно, которое должно обрабатываться клиентским приложением, используйте методы упаковки в OLE-сервере для метода Move и свойства упаковки для Text.

Совет 35. Будьте осторожны, когда завершаете работу OLE-серверов

Не используйте никаких внешних методов для завершения работы вашего сервера. Пусть сервер сам автоматически прекратит работу, когда к нему не останется ссылок. Если вы используете позицию меню Exit в OLE-сервере, который выводит на экран пользовательский интерфейс, просто выгрузите формы, открытые пользователем, и освободите все созданные экземпляры объекта.

Совет 36. Установите все переменные объекта равными Nothing (Пусто)

Установите все переменные объекта равными Nothing (Пусто), прежде чем заканчивать приложение. При выполнении оператора End в клиентском приложении OLE OLE-сервер выключается, и события Terminate, относящиеся к тем объектам, которые еще не прекратили работу, не выполняются.

Совет 37. Используйте самый конкретный тип объектов из всех имеющихся

Вместо того чтобы объявлять тип объекта как Object (As Object), используйте конкретный тип объекта, такой, например, как Ctask. Это повышает производительность за счет минимизации требований к перекодировке в OLE.

Совет 38. Используйте подстановку объектов

Используйте подстановку объекта, чтобы заменить простое имя на расширенную ссылку на объект. Каждый знак ".", используемый в коде, представляет собой перекодировку OLE. Большая производительность достигается при минимизации числа выполнения перекодировки. Например:

```
For i = 1 to 10
   txt(i) = myApp.TimeSheet.Employee.Name
Next i
```

Данный пример будет выполняться быстрее, если его переписать, используя подстановку:

```
Dim emp As New Employee
' Создание переменной для подстановки объекта
Set emp = myApp.TimeSheet.Employee
For i =1 to 10
   txt(i) = emp.Name
Next i
```

Андрей Колесов, Ольга Павлова

© 1996, Андрей Колесов, Ольга Павлова
Авторский вариант. Статья была опубликована с незначительной литературной правкой в журнале "КомпьютерПресс" N 9/96, с. 70-74.

Совет 39. Тестируйте различные варианты реализации своих программ с целью выбора оптимального

На эту тему мы уже говорили ранее, рассказывая об отдельных конструкциях языка. Но данная проблема довольно обширна, поэтому нелишне обратиться к ней снова. На этот раз поводом послужила подборка материалов, опубликованная в журнале VBPJ ь 6'96 и посвященная обсуждению тестирования различных версий VB (16-разрядные VB 3.0 и VB 4.0, 32-разрядный VB 4.0) в разных условиях (Win 3.x, 95, NT; 486 и Pentium; работа с графикой, базами данных, оконным интерфейсом и численная обработка).

Мы решили познакомить наших читателей с некоторыми выводами, сделанными в этих статьях. Но прежде несколько общих замечаний.

К любым результатам тестовых исследований нужно относиться весьма осторожно, поскольку нередко они зависят от конкретной методики тестирования, наборов исходных данных и пр. Конечно, есть некоторые рекомендации, привносящие уже на теоретическом уровне известный качественный результат. Например, прямая работа через функции API (а в DOS — через системные функции BIOS и DOS) будет, скорее всего, выполняться быстрее, чем через операторы языка. Но писать и отлаживать такую программу — дело более хлопотное.

Однако в большинстве случаев все обусловлено спецификой конкретного приложения, структурой базы данных и т.п. Кроме того, определенное влияние оказывают и версии используемого языка (что довольно часто связано с обыкновенными "плюхами" в его конкретной реализации). Причем это относится и к "теоретически очевидным" вещам (быстрее, но насколько?).

Здесь мы хотели бы подчеркнуть, что приведенные ниже выводы интересны не столько с точки зрения их абсолютной достоверности, сколько как хороший повод задуматься: а не проверить ли мне, как будет работать в моей программе другая конструкция? Не даст ли это некоторый положительный эффект?

Разумеется, любые исследования нужны только тогда, когда в них есть смысл. Если графический образ выводится на экран за 0,1 с, то вряд ли стоит бороться за 0,05 с. Но когда разовое обращение к базе данных выполняется за 10 с, то пользователь очень обрадуется, если время его ожидания сократится до 1 с.

Итак, некоторые выводы из статей VBPJ #6'96.

Статья 1: Clocking Data Access (Измерение времени доступа к данным) by Steve Jackson

Время выполнения запроса может варьироваться в соотношении 24:1 в зависимости от метода доступа к данным. Изучите результаты проведенных эталонных тестов для выбора необходимого вам подхода.

1. Непосредственный вызов функций API ODBC — самый быстрый метод. Эта скорость частично обусловлена обходом Jet и DAO (Data Access Object), благодаря чему высвобождаются 2 Мбайт памяти, используемые Jet и DAO. Однако функции API намного сложнее кодировать, и код API хуже поддерживается, чем код DAO.
2. RDO (Remote Data Object) — компромисс между ODBC API и Jet. Он намного быстрее, чем Jet и DAO, и его легче кодировать, чем ODBC API. (Но RDO существует только для 32-разрядной версии.)
3. Идентичные 16-разрядные программы в целом выполняются немного быстрее в Win95, чем в Windows for Workgroups 3.11. Такой результат объясняется более совершенной внутренней организацией Win95 и лучшим управлением памяти, более быстрым стеком TCP/IP или двумя этими факторами, вместе взятыми.
4. В Win95 32-разрядные приложения выполняются немного медленнее, чем 16-разрядные.
5. В WinNT в отличие от Win95 32-разрядные тестовые программы выполняются намного быстрее, чем 16-разрядные программы. Эта разница более существенна для методов с большими непроизводительными затратами и большим числом DLL-библиотек, например

таких, как метод Jet DAO, и менее существенна для методов с меньшими непроизводительными затратами, как ODBC API.

Причина: WinNT выполняет 16-разрядные программы как отдельные задачи, называемые Windows on Windows (WOW). Поскольку WOW преобразует все вызовы 16-разрядных функций в 32-разрядные команды, он медленнее, чем Win95 или Win 3.11, которые выполняют 16-разрядные вызовы непосредственно. В NT нет 16-разрядного кода, поэтому 32-разрядный код имеет огромное преимущество.

6. VB4 не оказывается существенно быстрее, чем VB3, для рассмотренных методов работы с БД. Большинство циклов в тестовых примерах выполняется во внутреннем компоненте (back end), а не в VB-коде, поэтому последний не является вентильным фактором (gating factor) для доступа к данным.
7. Модернизация аппаратных средств — не единственный метод повышения скорости доступа к данным. Решение проблемы — изменить код.

Статья 2: Benchmark Battle (Борьба эталонных тестов) by Ward Hitt

VB3 побеждает VB4 в графической обработке, однако в других тестах VB4 вырывается вперед как в 16-разрядном, так и в 32-разрядном режимах.

Ключевой вопрос: если вы перейдете в Win95 или WinNT, где установлена 32-разрядная версия VB4, будут ли ваши приложения выполняться быстрее?

Ответ: все зависит от типа приложения. Для графической обработки с помощью VB-методов VB4/32 — самый медленный, а для других операций — самый быстрый. Иногда VB4 имеет ту же скорость, что и VB3.

1. VB3 является победителем во всех графических тестах, если используются VB-методы. В Win 3.11 VB3 на 33% быстрее, чем VB4/16. В Win95 VB3 на 56% быстрее, чем VB4/16, и почти в 3 раза быстрее, чем VB4/32. VB3 убедительно лидирует и в WinNT. VB4 имеет плохие показатели в графической обработке, так как он должен подключаться (thunk) к 16-разрядному GDI.
2. Автор рекомендует избегать графических методов в VB и вместо этого использовать вызов графических функций API. Последние резко снижают время обработки графики на 486-х компьютерах и существенно улучшают время обработки на Pentium. VB4/16 лидирует во всех тестах, использующих вызов графических функций API, за исключением WinNT, где победил VB4/32.
3. Загрузка форм (используется во всех VB-приложениях). VB4/16 почти на 33% быстрее, чем VB3 под Win 3.11, на 15% быстрее в WinNT и медленнее как на 486, так и на Pentium в Win95. VB4/32 медленнее, чем VB4/16, загружает тестовую форму в Win95, но намного быстрее, чем VB3 или VB4/16, в WinNT. VB3 быстрее всех загружает форму в Win95.
4. Вывод текста в элементах управления. VB4/16 в 2 раза быстрее, чем VB3, в Win 3.11, почти равен в Win95, но медленнее в WinNT. VB4/32 почти на 25% медленнее других версий в Win95, но намного быстрее в WinNT.
5. Скорость числовых расчетов увеличивается от VB3 к VB4/16 и до VB4/32. VB4/16 быстрее, чем VB3, в Win 3.11/NT и равен в Win95. VB4/32 значительно быстрее, чем VB3 и VB4/16, за исключением конкатенации строковых переменных фиксированной длины (fixed-string). Эта операция выполняется крайне медленно в VB4/32 (второй недостаток VB!), а быстрее всего — в VB3. По неофициальным сведениям, полученным из Microsoft, данная операция представляет собой одну из целей для проведения усовершенствований в следующей версии VB.

6. WinNT был разработан Microsoft как более надежная и помехоустойчивая система по сравнению с Win95, хотя, по-видимому, при этом была принесена в жертву скорость вывода. Тем не менее, числовые расчеты (number crunching) выполняются быстрее в NT, чем в Win95.
7. При выполнении неграфических приложений на Pentium с большим объемом оперативной памяти разница в скорости между тремя версиями VB будет измеряться в долях секунды.
8. В целом версии VB4 примерно равны или лучше, чем VB3, для Win95 или NT. Если вы работаете в Win 3.11 с 8 Мбайт оперативной памяти, VB3 по-прежнему является хорошим средством разработки.

Совет 40. Минимизируйте количество повторяемых перекодировок OLE

Используйте оператор With...End With для минимизации количества повторяемых перекодировок OLE. Этот способ имеет дополнительное преимущество: он не требует временного объекта для подстановки.

Совет 41. Используйте in-process серверы при любой возможности

Для повышения производительности используйте in-process серверы (в виде библиотек OLE DLL), когда это только возможно. Выполнение вызовов объектов внутри пространства приложения — либо в самом приложении, либо в in-process сервере — осуществляется намного быстрее, чем вызовы объектов за пределами пространства приложения.

Совет 42. Как ускорить вызовы OLE-сервера

Передавайте как можно больше данных в/из OLE-сервера за каждую операцию вызова. Передача данных выполняется быстро, а операция вызова — медленно, особенно когда вы имеете дело с out- of-process серверами. Например, вы можете передать набор параметров в массиве, вместо того чтобы вызывать OLE-сервер несколько раз.

Совет 43. Инициализация DLL-библиотеки, представленной в виде in-process OLE-сервера

Запустите in-process OLE-сервер с помощью процедуры Sub Main. Включите в нее код инициализации всех серверов. Не выводите на экран формы из этой процедуры и не используйте Command Function для получения содержимого командной строки (она не существует при инициализации in-process сервера).

Совет 44. Тестирование DLL-библиотеки, представленной в виде in-process OLE-сервера

Чтобы протестировать DLL-библиотеку в виде in-process OLE-сервера, сначала постройте ее в виде out-of-process сервера. Это существенно упрощает операции отладки и тестирования. Имейте в виду, что in-process сервер также будет влиять на стабильность всего приложения. Поэтому хорошей идеей является обеспечение стабильности OLE-сервера в виде out-of-process сервера, а затем уже преобразование его в DLL-библиотеку OLE.

Совет 45. Тестируйте in-process OLE-сервер как out-of-process сервер

Используйте опцию OLE Restrictions для тестирования in-process OLE-сервера как out-of-process сервера. Данная опция находится во вкладке Advanced диалогового окна Options. Она активизирует ограничения DLL, хотя вы и используете out-of-process сервер. Такой подход эффективен при проверке работы сервера в качестве in-process сервера.

Совет 46. Как выгрузить сервер из памяти

Закройте экземпляр Visual Basic, который использует скомпилированный in-process OLE-сервер, чтобы выгрузить этот сервер из памяти. Изменения в in-process OLE-сервере не будут видны до тех пор, пока текущая версия сервера не будет выгружена из памяти, а новая загружена.

Совет 47. Как установить MousePointer в in-process OLE-сервер

Не устанавливайте MousePointer в библиотеку OLE DLL до тех пор, пока полностью не убедитесь, что это необходимо. In-process OLE-сервер не может осуществить поиск текущего курсора мыши, поэтому у вас нет возможности установить его обратно в то состояние, которое требуется приложению. Если все же необходимо установить MousePointer в in-process OLE-сервер, всегда устанавливайте его в значение по умолчанию перед тем, как вернуть в клиентское приложение. Если клиентское приложение имеет другой набор MousePointer, тогда оно должно вернуть MousePointer в необходимое состояние после обращения к какому-либо свойству или методу сервера.

Совет 48. Как передать массив элементов управления

Работа с массивами элементов управления в VB3 была сплошной нервотрепкой, но в VB4 появилась возможность передавать такой массив в качестве аргумента функции. Для этого необходимо просто указать тип параметра как Variant:

```
Private Sub Command1_Click(Index As Integer)
   GetControls Command1()
End Sub
Public Sub GetControls(CArray As Variant)
  Dim C As Control
  For Each C In CArray
    MsgBox C.Index
  Next
End Sub
```

Кроме того, массивы элементов управления в VB4 имеют свойства Lbound, Ubound и Count:

```
If Command1.Count < Command1.Ubound - Command1.Lbound + 1 Then_
     MsgBox "Массив не является непрерывным"
```

VB 5.0 — хорошие новости для России

В начале июня в Обнинске прошла очередная российская конференция Microsoft для разработчиков — DevCon'96. На ней была официально объявлена следующая информация. В версии 5.0 будет реализован настоящий транслятор, создающий не менее настоящие исполняемые модули на машинном языке! В VB 5.0 будут наконец-то

реализованы все основные элементы объектно-ориентированного языка, а не какие-то псевдообъекты в виде элементов управления и классов (то, как они понимались в VB 4.0). Как уже сообщалось ранее, у VB- программистов появится возможность создания OCX на этом же языке.

Версия VB 5.0 будет локализована для России! Работа по локализации будет вестись параллельно с общей подготовкой к новой версии (фактически — это этап начинающегося бета- тестирования), и локализованную версию планируется выпустить практически одновременно с основной, американской, осенью этого года.

Локализация будет проходить по схеме, уже применявшейся для других систем программирования (в частности, для Visual FoxPro): будет переводиться только документация и справочная система, программная же часть пакета останется неизменной. Но при этом в ходе доводки VB 5.0 до окончательного варианта особое внимание будет уделяться возможности работы VB с кириллицей. Это касается проблем ввода-вывода русских букв, сортировки символьных данных и пр.

Бета-тестирование уже началось. Сейчас в нем участвуют две российские фирмы (Microsoft не сообщает о конкретных участниках тестирования, как и об исполнителях локализации). На этот раз число бета-тестировщиков резко сокращено — это общая политика фирмы во всем мире. Через два месяца, на следующем этапе тестирования, число участников может быть расширено. По заявлениям представителей Microsoft, прозвучавшим не только на DevCon'96, но и в международной прессе, роль VB как стратегического направления средств разработки будет расти и в дальнейшем. В частности, фирма называет именно его, а также его подмножество VBScript в качестве основной системы, которая составит альтернативу Java. Уже известно, что многие будущие расширения VB связаны с работой в Internet.

Другая тенденция — расширение возможностей работы с базами данных. В частности, на DevCon'96 прозвучало следующее: Microsoft планирует прекратить развитие направления FoxPro как самостоятельного средства разработки и интегрировать его с Visual Basic. Тем не менее, этой осенью выйдет новая версия Visual FoxPro 5.0 (сразу после прошлогодней 3.0), возможно, через год-полтора появится еще одна версия, которая, скорее всего, будет последней. По крайней мере, таковы намерения Microsoft на сегодня.

Известие о такой судьбе FoxPro, естественно, взволновало его многочисленных отечественных пользователей. Однако представители Microsoft призывали их не огорчаться, выдвигая следующие аргументы:

- это только планы, которые могут и не осуществиться; в любом случае у разработчиков FoxPro есть еще в запасе 4-5 лет для работы в этой системе;
- фирма сделает все возможное, чтобы обеспечить безболезненный переход от FoxPro к VB.

Заметим, что старые версии систем разработки действительно не умирают с выходом новых. По некоторым оценкам, большинство пользователей FoxPro (причем не только для России) продолжают работать с версиями 2.x, хотя Visual FoxPro появился еще два года назад. С другой стороны, опыт показывает, что любые планы могут корректироваться в обоих направлениях — скорее версия 6.0 не будет выпущена вообще, чем появится 7.0. В любом случае очевидно, что разработчикам FoxPro есть над чем подумать. Что же касается VB-программистов, то их такая информация заметно обрадовала, так как она свидетельствует о хороших перспективах развития этой системы, которая в настоящее

время в некоторых вопросах даже обычного программирования (например, использования объектно- ориентированных средств) значительно уступает Visual FoxPro.

Андрей Колесов, Ольга Павлова

© 1996, Андрей Колесов, Ольга Павлова
Авторский вариант. Статья была опубликована с незначительной литературной правкой в журнале "КомпьютерПресс" N 10/96, с. ??-??.

Совет 49. Устанавливайте текст в элемент управления Masked Edit программным образом

Элемент управления Masked Edit имеет свойство Text, но оно не отображается в окне Properties, то есть начальную установку текста нельзя выполнять в процессе разработки программы. Поэтому такую установку нужно выполнять на программном уровне, например в момент загрузки формы:

```
Private Sub Form_Load ()
   MaskEdBox1.Text="369-76-97"
End Sub
```

При работе с Masked Edit следует помнить, что строковая переменная должна полностью совпадать с символами в шаблоне ввода (свойство Mask), включая литерные символы и подчеркивание. В приведенном выше примере подразумевалось, что шаблон номера имеет вид "###-##-##".

Совет 50. Традиционный: используйте самые быстрые конструкции

1. Чтобы переместить программным образом элемент управления или форму в новое место, можно просто изменить значения свойств Left и Top, например:

```
2.   frmCustomer.Left = frmCustomer.Left + 100
3.   frmCustomer.Top = frmCustomer.Top + 50
```

   Но, используя метод Move, можно сделать то же самое процентов на 40 быстрее:

```
frmCustomer.Move = frmCustomer.Left + 100, _
frmCustomer.Top + 50
```

4. Если вам нужно сделать проверку числовой переменной с нулем, то логическая конструкция

```
5.   If iNumber Then
```

   будет работать быстрее, чем арифметическая

```
If iNumber <> 0 Then
```

Совет 51. Модифицируйте (если это нужно) режим Tab при вводе данных

При вводе данных в текстовые поля формы бывает не очень удобно пользоваться клавишей Tab. Например, вы привыкли, что ввод данных в текущее поле заканчивается нажатием Enter. Чтобы сделать это, добавьте такой код для нужного элемента управления:

```
Sub Text1_KeyPress (KeyAscii As Integer)
  If KeyAscii = 13 Then   ' клавиши Enter
    SendKeys "{Tab}"
    KeyAscii = 0
  End If
End Sub
```

А чтобы реализовать режим AutoTab при вводе данных в тестовое поле, можно использовать проверку свойства MaxLength в процедуре обработки события Change:

```
Sub Text1_Change ()
  If Len(Text1)=Text1.MaxLength Then
     SendKeys "{Tab}"
  End If
End Sub
```

В этом случае переход к следующему полю будет выполняться автоматически после ввода последнего символа.

Совет 52. Простой способ переключения флагов

В программах довольно часто приходится использовать переменные-флажки, которые имеют значения "ноль/не ноль". Конечно, можно написать:

```
If bPerform Then bPerform = False Else bPerform = True
```

Но так будет выглядеть симпатичнее:

```
bPerform = Not bPerform    ' состояние флага 0 или -1
```

А если вы больше привыкли иметь дело с арифметическими операциями, то можно использовать операцию Xor:

```
iFlag = iFlag Xor iMaskFlag
```

Последний вариант интересен тем, что в этом случае может использоваться любое ненулевое значение флага, которое в этом случае представлено iMaskFlag (1, -1, 777 и пр.). Еще одно замечание. Будьте внимательны, используя оператор Not при работе с целочисленными переменными, которые могут принимать значения, отличные от -1 (True) и 0 (False):

```
Sum cmdBool_Click()
  Dim iBool As Integer, iTemp As Integer
  iBool = True
  Print iBool     ' печатается -1
  Print Not iBool ' печатается 0
  iTemp = 5
  Print iTemp     ' печатается 5
  Print Not iTemp ' печатается -6
  ' любое ненулевое значение воспринимается как True
  If iTemp Then
    Print "iTemp = True" ' печататься будет здесь Else
    Print "iTemp = False"
  End If
End Sub
```

Совет 53. Используйте статические переменные (когда нужно)

Все переменные (в том числе и массивы) в VB могут быть динамическими или статическими. Их принципиальное отличие заключается в том, что резервирование и освобождение динамических переменных осуществляется в процессе выполнения программы по некоторым специальным запросам или автоматически при выполнении определенных операций. Например, при обращении к процедуре автоматически резервируются все ее локальные динамические переменные, а при выходе из нее они так же автоматически освобождаются. Для простых переменных это реализуется в виде некоторого стека, а для массивов — довольно сложным алгоритмом работы с динамической памятью. Но программисту особого дела до этого нет, так как здесь работает непосредственно VB. Хотя иногда при разработке сложных проектов приходится учитывать особенности механизма диспетчеризации (это бывает, когда вдруг появляются неприятные сообщения типа Out of memory).

Статические переменные формируются на этапе компиляции и существуют в процессе работы программы.

Достоинства и недостатки эти типов переменных понятны: динамические переменные позволяют оптимизировать использование памяти — они создаются только на время их реальной необходимости. Но работа с ними требует несколько большего времени для выполнения операций резервирования и освобождения. Хотя последнее замечание — чисто теоретическое: несколько моих попыток отловить на тестах эти временные отличия не привели к сколь-нибудь значимым результатам.

В VB по умолчанию все локальные переменные любой процедуры являются динамическими. Естественно, что в результате этого все значения локальных переменных теряются после выхода из нее. Поэтому, если вы хотите сохранить содержимое переменных между обращениями к процедуре, следует объявить их статическими, заменив ключевое слово Dim на Static. Это особенно полезно при создании постоянных счетчиков, а также в режиме отладки.

Приведем пример, демонстрирующий общее число одиночных щелчков мыши по кнопке (первый раз, когда вы щелкаете кнопку, счетчик начинает счет со значения по умолчанию, равного нулю):

```
Sub Command1_Click()
  Static Counter As Integer   ' Счет начинается с 0
  Counter = Counter + 1
  Print Counter
End Sub
```

Обратите внимание, что все переменные, описанные как глобальные на уровне модуля (оператором Dim в разделе Declarations), являются также статическими. Используя их, можно легко создать общие счетчики для нескольких процедур, например так:

```
'  раздел Declarations:
Dim Counter As Integer
' Counter - общий счетчик щелчков по двум кнопкам

Sub Command1_Click()
  Counter = Counter + 1
  Print Counter
End Sub
```

```
Sub Command2_Click()
  Counter = Counter + 1
  Print Counter
End Sub
```

А если вы хотите, чтобы отсчет начинался с нуля при каждой загрузке формы, можно добавить следующий код:

```
Sub Form_Load ()
  Counter = 0
End Sub
```

Для описания глобальных переменных на уровне модуля в VB 4.0 можно использовать оператор Private, а для создания переменных, доступных в любых процедурах всего приложения, — Public.

Совет 54. Будьте внимательны при работе с динамическими массивами

Динамические массивы представляют особую ценность для программиста. Они позволяют резервировать размеры массива, соответствующие реальным требованиям задачи, меняя их в случае необходимости (здесь есть интересные моменты, связанные с возможностью сохранения данных при перерезервировании массивов). В DOS'овских версиях они также позволяли использовать свободную основную память компьютера (статические массивы вместе с простыми переменными размещались в ближнем сегменте данных 64 Кбайт).

Динамические массивы можно создавать как на уровне процедуры, так и на уровне модуля. В первом случае они создаются и существуют, как и простые локальные переменные процедуры, только на время ее выполнения (с момента выполнения оператора Dim до Exit Sub/Function). Во втором случае они становятся как бы псевдостатическими: хранятся постоянно в памяти, но управление их резервированием (изменением размерности) выполняется в явном виде с помощью операторов ReDim в процедурах данного модуля. Однако при объявлении динамических массивов на уровне модуля есть некоторые нюансы, на которые следует обратить внимание.

Но прежде чем рассмотреть их для VB/Win, имеет смысл вспомнить, как это выглядело для DOS'овских версий (Quick, PDS, Visual). Общими правилами здесь являются следующие:

- в ходе работы программы нельзя менять тип данных (например, SINGLE вместо INTEGER) и число индексов (то есть если определен двухмерный массив, то перерезервировать его в одномерный уже нельзя);
- для фактического создания массива нужно выполнение оператора REDIM в явном виде.

Рассмотрим такой пример для Basic/DOS:

```
' Модуль TEST1.BAS
' описание типа и размерности динамического массива,
' глобального на уровне модуля:
REDIM SHARED Array(10,20) AS INTEGER

SUB TestSub1
  PRINT Array(3,3)
  REDIM Array(8,97) AS INTEGER ' перерезервирование массива
END SUB
```

```
SUB TestSub2
  REDIM Array(4,7) AS INTEGER  ' начальное создание массива
  PRINT Array(2,2)
END SUB
```

Оператор REDIM SHARED... в разделе объявлений говорит о том, что определяется целочисленный двухмерный массив, глобальный на уровне модуля (для этого используется слово SHARED). Но фактически никакого массива не создается (имеется в виду, что это не головной модуль программы, с которого начинается ее выполнение). Описание границ массива (10,20) игнорируется и нужно только для того, чтобы показать число его индексов. Поэтому если мы сразу обратимся к процедуре TestSub1, то при выполнении оператора PRINT Array(3,3) будет выдано сообщение об ошибке ь 9 (хотя в нем говорится о нарушении границ, на самом деле массива просто не существует). Для работы с массивом нужно его создать, обратившись, например, вначале к процедуре TestSub2, а уже после этого вызывать TestSub1, в которой в том числе можно изменять его границы.

Обратите внимание, что при работе в среде QB (только в ней!) при определении массива в процедуре можно опустить описание типа AS INTEGER, но массив все равно будет резервироваться целочисленным. Впрочем, компилятор в любом случае выдаст ошибку о несовпадении типов данных и заставит вас устранить эту двусмысленность.

Если в процедуры этого же модуля добавить такие операторы, то они будет сразу определены как ошибочные:

```
REDIM Array(100)        ' Изменение числа индексов массива
REDIM Array(4,7) AS SINGLE ' Изменение типа данных
DIM Array(4,7) AS INTEGER ' Дубликатное резервирование массива:
' имя локального массива совпадает с
' глобальным на уровне модуля
```

Теперь посмотрим, как подобная конструкция может выглядеть для VB/Win 4.0:

```
'  описание ТОЛЬКО типа (целочисленного) динамического массива,
'  глобального на уровне модуля:
Dim Array() As Integer

Private Sub Form_Load()
  Dim Array(10)   ' это локальный динамический массив!!!
  Array(3) = 15.5
  Debug.Print Array(3)   ' к тому же - Variant
  ' (будет напечатано 15.5)
End Sub

Private Sub Command1_Click()
  Print Array(3)  ' будет напечатано 3
  ' или появится сообщение об ошибке
End Sub

Private Sub Command2_Click()
  ' резервируем двумерный массив
  ReDim Array(10, 20)
  Array(2, 2) = 22.3   ' но Integer!
  Print Array(2, 2)    ' будет напечатано 22
  ' резервируем одномерный массив
  ReDim Array(15)
  Array(3) = 3.3
```

```
  Print Array(3) ' будет напечатано 3
End Sub

Private Sub Command3_Click()
  ' резервируем трехмерный массив
  ReDim Array(10, 20, 30)
  Array(1, 2, 3) = 123.1
  Print Array(1, 2, 3)
End Sub

Private Sub Command4_Click()
  ' пытаемся изменить тип данных
  ReDim Array(10, 20) As Single
End Sub
```

Прежде всего следует отметить, что запуск такого приложения в среде VB/Win 4.0 выполняется без проблем — никаких ошибок синтаксиса не обнаружено (Basic/DOS тут покажется чрезмерно строгим). Здесь следует обратить внимание на следующее:

- для описания динамического массива в Declarations используется оператор Dim с пустыми скобками (). Уже одно это говорит, что нет фиксации числа индексов — их можно менять в ходе работы (см. процедуры Command2_Click и Command3_Click);
- допускаются дубликатные имена глобальных и локальных массивов. В процедуре Form_Load оператор Dim Array(10) резервирует ЛОКАЛЬНЫЙ динамический массив (к тому же типа Variant), который не имеет никакого отношения с глобальному, имеющему то же имя. Если после запуска программы (то есть выполнения процедуры Form_Load) сразу щелкнуть кнопку Command1, то появится сообщение о том, что массив Array не определен;
- перерезервирование массива с изменением числа индексов вполне возможно, причем даже в одном модуле (Command2_Click). Но естественно, что определенный порядок при этом должен соблюдаться. Здесь может возникнуть путаница, если операторы резервирования и обращения к массиву находятся в разных процедурах (как это сделано в данном примере). Так, после щелчка кнопки Command2 можно без проблем щелкать и Command1. Но после Command3 щелчок Command1 уже вызовет ошибку;
- тип перерезервируемых в процедурах массивов в явном виде можно не указывать (см. Command2_Click и Command3_Click) — он будет все равно целочисленным. Никакой диагностики не появится и на этапе создания исполняемого модуля. Более того, при запуске приложения в среде не будет никакой реакции на попытку изменения типа массива в процедуре Command4_Click(). Диагностика ошибки появится только при щелчке кнопки Command4 или на этапе компиляции при создании исполняемого модуля.

Некоторые выводы

1. К работе с глобальными динамическими массивами в VB нужно относиться очень внимательно.
2. Потенциально использование дубликатных имен несет в себе определенную угрозу путаницы. Например, возможно, в приведенном выше фрагменте мы просто ошиблись, написав в процедуре Form_Load оператор Dim вместо ReDim, и при этом уверены, что имеем дело с глобальным массивом. На уровне модуля и его отдельных процедур противоречий между именами локальных и глобальных переменных не должно быть (ведь модуль пишется одним человеком). Реально это может пригодиться только для устранения возможных конфликтов между идентификаторами переменных, определенных на уровне модуля (Private) и на уровне приложения (Public). Хотя для разрешения подобных конфликтов классические принципы языков программирования требуют описания глобальных переменных не только в модулях, где они резервируются,

но и в модулях, где они используются (в качестве входных точек). К сожалению, в VB этого механизма почему-то нет.

3. Зачем нужно переопределять число индексов массива, просто не приходит в голову (такая возможность появилась только в версии 4.0). По нашему мнению, кроме путаницы, это ничего не приносит. Если у читателей есть хорошие примеры практического использования такого механизма, то было бы очень интересно узнать об этом. Кстати, реализованный в Basic/DOS механизм определения числа индексов массива на уровне Declarations представляется наиболее удобным. В VB 3.0 просто выдается сообщение о том, что имеет место противоречие описания массива с каким-то другим описанием, которое находится неизвестно где. А в Basic/DOS все подобные сравнения выполняются с известным главным описанием массива.

Совет 55. Будьте внимательны при переходе из Win16 в Win32

Проблем здесь довольно много, и одна из самых главных — преобразование 16-разрядного интерфейса API в 32-разрядный. В общем случае все обращения к функциям API нужно внимательно переделывать (это в первую очередь относится к операторам Declare). Здесь существуют несколько разных вариантов или их комбинаций:

- изменяется имя библиотеки, в которой находится функция (это обязательно);
- изменяется тип процедуры: вместо подпрограммы (Sub) используется функция (Function);
- изменяется название процедуры;
- изменяется число и состав параметров;
- изменяется тип числовых параметров;
- процедуры API, имеющиеся в Win16, в Win32 просто исчезли или заменены принципиально иными способами выполнения тех или иных операций.

При этом особое внимание требуется тогда, когда внешний вид обращений остался вроде бы тем же, например целые данные просто превратились из 16-разрядных в 32-разрядные (это относится и к кодам ошибок). Вот как, например, будет выглядеть одновременная реализация процедуры GetWindowRect в Win16 и Win32:

```
 #If Win32 Then
    Private Declare Function GetWindowRect Lib "user32" _
            (ByVal lHwnd As Long, uRect As RectType) As Long
    Private Type RectType
            Left As Long
            Top As Long
            Right As Long
            Bottom As Long
    End Type
 #Else
    Declare Sub GetWindowRect Lib "user" _
            (ByVal iHwnd As Integer, uRect As RectType)
    Private Type RectType
            Left As Integer
            Top As Integer
            Right As Integer
            Bottom As Integer
    End Type
 #End If
Dim uRect As RectType

Private Function lGetWindowRect(iHwnd As Integer, uRect As RectType)
  #If Win32 Then
     lHwnd = iHwnd
     lGetWindowRect = GetWindowRect(lHwnd, uRect)
```

```
  #Else
     lGetWindowRect = 0
     Call GetWindowRect(iHwnd, uRect)
  #End If
End Function
```

Проблема одновременного сопровождения 16- и 32-разрядного вариантов одного приложения несколько упрощается с появлением в VB 4.0 директив условной компиляции, хотя даже из этого простого примера видно, что хлопот с настройкой кода на работу и в Win16, и в Win32 вполне хватает. В данном случае API-процедура из подпрограммы превратилась в функцию, а первый передаваемый параметр поменял тип с Integer на Long. Еще одно отличие, которое не сразу заметно: поля структуры uRect также стали Long.

Очевидно, что дело здесь сводится не только к замене описания Declare, но и к коррекции кода программы, где выполняется работа с процедурой и ее параметрами.

Что касается определяемых пользователем структур, то следует иметь в виду, что в VB выполняется выравнивание полей. Например, длинное целое будет иметь адрес, кратный 4. Это надо учитывать при работе с библиотеками, использующими иной способ упаковки.

Довольно неприятным моментом является и то, что в Win32 имена функций стали чувствительными к регистру букв. Если в приведенном выше примере в операторе Declare вы напишите имя GetwindowRect, то во время исполнения приложения при обращении к этой функции появится сообщение об ошибке "Specified DLL function not found" (запрошенная DLL-функция не найдена). Но за соответствием идентификаторов в Declare и в последующем коде программы VB следит сам.

В заключение этой темы имеет смысл напомнить, что с помощью ключевого слова Alias в операторе Declare можно менять фактические имена DLL-функций на альтернативные. Это может, например, пригодиться, если имена аналогичных API функций в версиях Win16 и Win32 отличаются.

Разумеется, здесь рассмотрены далеко не все проблемы перехода из Win16 в Win32, в том числе в плане использования функций API. По довольно единодушному мнению различных экспертов, самым надежным и лучшим пособием в этом вопросе является книга известного американского автора и разработчика VB-продуктов Дэна Эпплмана (Daniel Appleman) "Visual Basic Programmer's Guide to the Win32 API" (издательство Macmillan Computer Publishing/Ziff-Davis Press, 1996). Ранее его же книга, посвященная Win16 API, была бестселлером в течение нескольких лет.

Для тех, кто занимается преобразованием программ из VB 3.0 в VB 4.0, можно посоветовать использовать бесплатную копию комплекта программ и документации Upgrade Wizard подразделения Crescent фирмы Progress Software, которую можно найти на Web-странице: http://crescent.progress.com. Там содержится много ценной информации и полезных рекомендаций, а утилита-"мастер" автоматически преобразует значительную часть кода, расставив пометки там, где нужно исправить его вручную. Причем с ее помощью можно получить код как для Win16, так и для Win32.

Совет 56. Как определить, с каким приложением вы работаете в VBA

Когда вы создаете вспомогательные программные модули, предназначенные для использования в разных приложениях, которые могут быть как 16-, так и 32-разрядными, то в них нужно уметь определять тип приложений.

При работе с VBA и Access проверку режима работы приходится выполнять непосредственно в процессе выполнения приложения. Приведенные ниже примеры процедур используются для определения того, является ли конкретное приложение 32-разрядным или нет. Обратите внимание, что свойство Application.OperatingSystem в Microsoft Excel и в Microsoft Project возвращает не установленную версию Windows, а слой Windows, на котором выполняется приложение, например 16-разрядная подсистема в Windows NT.

1. Microsoft Excel 5.0, Microsoft Excel 95, Microsoft Project 4.0 и Microsoft Project 95:

```
Function Engine32%()
  If instr(Application.OperatingSystem, "32") Then Engine32% = True
End Function
```

5. Word 6.0 или Word 95:

```
Function Engine32
  If Val(GetSystemInfo$(23)) > 6.3 Or Len(GetSystemInfo$(23)) = 0 _
    Then Engine32 = -1 Else Engine32=0
End Function
```

10. Microsoft Access 1.x:

```
Function Engine32%()
  If SysCmd(7) > 2 Then Engine32% = True
End Function
```

Вот пример работы, связанный с "разрядностью":

```
Declare Function GetTickCount32 Lib "KERNEL32" _
  Alias "GetTickCount" () As Long
Declare Function GetTickCount16 Lib "USER" Alias _
  "GetTickCount" () As Long

Function GetTickCount() As Long
  If Engine32%() Then
     GetTickCount=GetTickCount32()
  Else
     GetTickCount=GetTickCount16()
  End If
End Function
```

Здесь следует обратить внимание на то, что в Win16 и Win32 используемая API-функция GetTickCount имеет одно и тоже название. Поэтому, когда нет возможности применить механизм условной компиляции, необходимо использовать управляющее слово Alias для того, чтобы изменить имя функции по крайней мере в одном из операторов Declare (в данном примере GetTickCount32 и GetTickCount16). Затем в зависимости от разрядности приложения переменная GetTickCount устанавливается в соответствии с точным именем функции API (GetTickCount32 или GetTickCount16) и вызовом данной функции API.

Андрей Колесов, Ольга Павлова

© 1996, Андрей Колесов, Ольга Павлова
Авторский вариант. Статья была опубликована с незначительной литературной правкой в журнале "КомпьютерПресс" N 11/96, с. 85-89.

Совет 57. Используйте функцию Len для проверки пустой строковой переменной

Как уже подчеркивалось ранее, все операции над строковыми переменными, в том числе сравнения, выполняются существенно медленнее операций с числовыми данными и даже часто уступают по времени обращения к числовым функциям. В частности, при проверке пустой строки конструкция:

```
If Len(sName) Then ...
```

выглядит предпочтительнее, чем:

```
If sName <> "" Then ...
```

Совет 58. Как сделать невидимыми на форме все ее элементы управления

Для этого следует просмотреть весь список элементов управления и присвоить значение False каждому свойству Enabled с помощью такого кода:

```
Sub cmdArray_Click ()
  Dim iLoop As Integer
  For iLoop = 0 To Me.Controls.Count - 1
    Me.Controls(iLoop).Enabled = False
  Next iLoop
End Sub
```

Так же просто сделать невидимой всю форму:

```
Me.Enabled = False
```

Однако следует помнить, что в последнем случае, сделав невидимым управляющее меню формы, вы сможете закрыть эту форму, лишь выгрузив ее из другой формы.

Совет 59. Быстрый поиск строки в Combo Box

Для быстрого поиска нужной строки в списке, в частности в элементе управления Combo Box, можно использовать функцию API SendMessage() (вариант для 16-разрядных приложений), обеспечивающую более эффективное выполнение программы, чем аналогичный код на VB. Для этого можно, например, прибегнуть к процедуре CBFindString:

```
Declare Function SendMessage Lib "User" _
  (ByVal hWnd As Integer, ByVal wMsg As Integer, _
  ByVal wParam As Integer, lParam As Any) As Long

Sub CBFindString(ctlEdit As Control, sSearch As String)
```

```
  Dim lPos As Long
  Const CB_FINDSTRING = &H40C
  lPos = SendMessage(ctlEdit.hWnd, CB_FINDSTRING, 0, ByVal sSearch)
  If lPos >=0 Then ctlEdit.ListIndex = lPos
End Sub
```

Совет 60. Как выгрузить все формы

Это можно сделать с помощью такой процедуры:

```
Public Sub UnLoadAll ()
  Dim i As Integer
  i = Forms.Count   ' количество загруженных форм
  Do While i > 0
    Unload Forms(i - 1)
    ' если форма не выгрузилась - завершить выполнение функции
    If i = Forms.Count Then Exit Do i = i - 1
  Loop
End Sub
```

Совет 61. Используйте для ожидания функцию API Sleep

При работе в Windows 95/NT для ожидания истечения некоторого отрезка времени (реализации задержки) вместо оператора DoEvent удобнее использовать функцию API (Win32) Sleep, которая должна быть описана следующим образом:

```
Public Declare Sub Sleep Lib "kernel32" _
  Alias "Sleep" (ByVal lMilliseconds As Long)
```

Обращение к ней может выглядеть так:

```
Sleep 15*1000&  'миллисекунды, в данном случае - 15 секунд
```

Совет 62. Программное определение режимов "Run Time" и "Design Mode"

При разработке VB-приложения иногда бывает полезно на уровне программного кода определить, в каком режиме это приложение работает: "Design Mode" (работа в среде VB) или "Run Time" (выполнение автономного модуля). Для этого, в частности, можно анализировать имена каталогов, где хранятся VB-проект и законченное приложение, примерно в таком виде:

```
If InStr (App.Path, "VB") Then Stop
```

Совет 63. Как загрузить форму VB 4.0 в среде VB 3.0

Форму, сделанную в VB4, нельзя напрямую использовать в среде VB3. Если это все же необходимо, следует вручную отредактировать файл с определением формы, который выглядит примерно так:

```
VERSION 4.00
Begin VB.Form Form1
Caption = "Form1"
ClientHeight = 5940
...
End
Attribute VB_Name = "Form1"
```

```
Attribute VB_Creatable = False
Attribute VB_Exposed = False
Option Explicit
```

Чтобы загрузить эту форму в VB3, нужно заменить VERSION 4.00 на VERSION 2.00, удалить все префиксы "VB.", которые имеются в конструкции Begin VB.Form Form1, а также удалить все объявления Attribute. Запомните этот файл и загрузите его в VB3.

Совет 64. Проверка существования файла

Для этого проще всего использовать команду Dir$:

```
FileName$ = "C:\WINDOWS\WIN.INI"  'имя проверяемого файла
  If Dir$ (FileName$) <> "" Then
        ' данный файл существует
  Else
        ' данный файл не существует
  End If
```

Совет 65. Создание многоуровневых каталогов

Иногда приходится анализировать наличие указанного каталога (к примеру, при установке вашей программы на жесткий диск) и создавать новый при его отсутствия. Для этого можно использовать, например, такую процедуру:

```
Sub CreateLongDir(sDir As String)
  Dim sBuild As String, sDirTmp As String, i As Integer
  '
  sDirTmp = sDir & "\"
  i = InStr (sDirTmp, ":")
  If i > 0 Then    ' задано имя диска
    sBuild = Left$(sDirTmp, i)      ' имя текущего каталога
    sDirTmp = Mid$(sDirTmp, i + 1)
  Else
    sBuild = ""    ' имя текущего каталога
  End If
  Do ' проверка-создание вложенных каталогов
    i = InStr (2, sDirTmp, "\")
    If i = 0 Then Exit Do
    sBuild = sBuild & Left$(sDir, i - 1)
    sDirTmp = Mid$(sDirTmp, i)
    If Dir$(sBuild, 16) = "" Then   'нет такого каталога
      MkDir sBuild    ' создание каталога
    End If
  Loop
End Sub

Sub Test () ' примеры обращения
  ' полное имя каталога с именем диска
  Call CreateLongDir("C:\Tests\TestDir\NewDir")
  ' полное имя каталога в текущем диске
  Call CreateLongDir("\Current\TestDir\NewDir")
  ' имя нового каталога относительно текущего каталога
  Call CreateLongDir("Current\TestDir\NewDir")
End Sub
```

Здесь крайне важно дать правильное описание имени каталога при обращении к CreateLongDir (в соответствии с правилами обращения к функциям VB: MkDir, ChDir, RmDir, Dir):

- в конце имени не должно быть символа "\";
- при указании имени диска между ним и именем каталога должен присутствовать символ "\";
- правильные имена: C:, C:\Test, \Test;
- неправильные имена: C:\, C:Test, Test\.

Совет 66. Редактирование ячеек Grid

Содержимое ячеек элемента управления Grid, поставляемого с VB, нельзя редактировать напрямую. Но проблема легко решается с помощью следующей конструкции:

```
Private Sub Grid1_KeyPress(KeyAscii As Integer)
  Grid1.Text = Grid1.Text & Chr(KeyAscii)
End Sub
```

Эта операция выполняется для текущей выбранной ячейки таблицы.

Совет 67. Как сделать Beep в VB4

Переход из VB3 в VB4 сопровождается некоторыми не очень приятные моментами, связанными с тем, что вроде бы очевидные программные конструкции работают в нем по-другому. Например, могут возникнуть проблемы с тривиальным гудком. Сколько раз "бибикнет" такая конструкция?

```
Beep: Beep
```

Ответ: в Basic/DOS, VB3 - два раза, а в VB4 - один.

Если на слух эта разница не очень убедительна, попробуйте другую конструкцию:

```
Beep:
Beep: A = 1
```

В этом случае при работе в VB4 вы вообще ничего не услышите. Почему-то зарезервированное слово Beep (так же, как и Cls), написанное первым в строке с разделителем ":", в VB4 воспринимается как метка.

Совет 68. Простой способ очистки выбранных элементов списка

В VB3 отменить выделение всех элементов списка можно таким образом:

```
List1.Selected (-1) = False
```

К сожалению, в VB4 эта конструкция не работает.

Совет 69. Перемещение элементов списка

Изменить расположение элементов списка можно, перетаскивая их мышью с помощью такой конструкции:

```
Sub List1_MouseDown (Button As Integer, Shift As Integer, _
  X As Single, Y As Single)
  ' Эта процедура запоминает номер текущего
```

```
  ' элемента списка и его содержимое
  ' при нажатии клавиши мыши
  Old_Index = List1.ListIndex
  TmpText = List1.Text
End Sub

Sub List1_MouseUp (Button As Integer, Shift As Integer, _
  X As Single, Y As Single)
  ' Когда вы отпустите клавишу мыши,
  ' указанный ранее элемент переместится
  ' в текущую позицию списка
  New_Index = List1.ListIndex
  If New_Index <> Old_Index Then
    List1.RemoveItem Old_Index   ' удалить старый
    List1.AddItem TmpText, New_Index ' вставить новый
  End If
End Sub
```

В общем модуле программы нужно зарезервировать глобальные переменные:

```
Dim TmpText As String
Dim Old_Index As Integer
Dim New_Index As Integer
```

Совет 70. Как посмотреть методы и свойства объекта

Чтобы подробнее познакомиться с составом и описанием методов и свойств различных объектов (например, элементов управления OCX), имеющихся на вашем компьютере, в VB 4.0 можно воспользоваться командой Object Browser:

1. Запустите VB. Нажмите F7, чтобы перейти в режим просмотра кода, а затем F2 - для запуска команды просмотра объектов.
2. На появившемся окне Object Browser в раскрывающемся меню Libraries/Projects выберите нужный файл (библиотеку или приложение). В списке Classes/Modules выделите интересующий вас объект, при этом в списке Methods/Properties появится описание его свойств и методов.
3. Чтобы познакомиться с каким-то свойством или методом, выделите его в списке - краткое описание будет выведено в нижней части окна. Для получения более подробной информации нужно щелкнуть кнопку помощи "?" на этом же окне.

Совет 71. Определение типа элемента управления

Если для продолжения процедуры необходимо определить, с каким типом элемента управления она в данный момент работает, можно применить следующую конструкцию, использующую функцию TypeOf:

```
Function myFunc (ctl As Control)
  ' Эта процедура работает для VB3 и VB4
  If TypeOf ctl Is TextBox Then
     ' Текстовое поле
  Else If TypeOf ctl Is CommandButton Then
     ' Кнопка
  End If
End Function
```

В VB4 это можно сделать с помощью новой функции TypeName, при этом код будет выглядеть примерно так:

```
Function myFunc (ctl As Control)
  Dim sCtlType As String
  sCtlType = TypeName(ctl)
  Select Case sCtlType
    Case "TextBox"
            ...
    Case "CommandButton"
            ...
  End Select
End Function
```

Узнать точное имя типа или класса конкретного элемента управления можно, посмотрев его в окне Properties в среде VB.

Совет 72. Как преобразовать приложения для Access 2.0 в приложения для Access 7.0

Приложение A в руководстве "Building Applications with Microsoft Access for Windows 95" содержит основные инструкции по преобразованию 16-разрядных приложений для Access 2.0 в 32-разрядные приложения для Access 7.0. Ниже приводится пошаговая инструкция преобразования для незащищенных баз данных, основанных на последней бета-версии Access 7.0. Она немного дополняет информацию, содержащуюся в руководстве:

1. Если ваше приложение для Access 2.0 вызывает дополнительные 16-разрядные DLL-библиотеки или использует 16-разрядные элементы управления OLE, убедитесь, что у вас есть соответствующие 32-разрядные версии.

2. Преобразуйте библиотеки или дополнительные средства, необходимые для Access 7.0. Вам, вероятно, потребуется обновить библиотеки или дополнительные средства третьих фирм на 32-разрядные версии.

3. Если вы еще не разделили свое приложение для Access 2.0 на отдельное приложение и файлы данных data.mdb, то сделайте это сейчас. Вы можете связать таблицы Access 2.0 с приложениями как для Access 2.0, так и Access 7.0, а не наоборот. Access 7.0 имеет ограничение на 32 индекса плюс связи для одной таблицы, выйдя за которое, то вы потеряете индексы и/или связи.

4. Удалите ненужные объявления функций Windows API, а также вызовы функций. Например, библиотека CTL3DV2.DLL не нужна в приложениях, выполняемых только в Windows 95.

5. Откройте модуль кода, затем из меню Run выберите команду Compile Loaded Modules, чтобы скомпилировать весь код, содержащийся в приложении. Если вы не сделаете этого, переход к Access 7.0 может не удасться.

6. Временно отключите макрокоманду AutoExec для своего приложения, предварительно переименовав ее.

7. Сожмите MDB-файл, используемый в вашем приложении.

8. Попытайтесь преобразовать свой MDB-файл. Если преобразование не получится, создайте новую базу данных в Access 7.0 и вручную импортируйте туда объекты из MDB-файлов Access 2.0, используя в меню File команду Get External Data, а затем Import.

9. Используйте Add-in Manager для подключения необходимых дополнительных средств к своему приложению.

10. Если ваше приложение обращается к библиотечному коду, выберите команду References из меню Tools для того, чтобы открыть диалоговое окно References. Затем установите тип .MDA в поле Files of Types, выберите необходимый библиотечный MDA-файл и создайте ссылку на него.

11. Если ваше приложение содержит элементы управления OLE, проверьте, что необходимые ссылки установлены для 32-разрядных, а не для 16-разрядных версий.

12. Откройте преобразованный модуль и выберите команду Compile All Modules из меню Run. Исправьте ошибки в программе таким образом, чтобы все модули компилировались без ошибок. Приложение A руководства описывает наиболее часто требуемые изменения, которые следует внести в код.

13. После того как вы скомпилировали все свои модули, выберите команду Save All Modules из меню File.

14. В окне Module Options установите параметр Break On All Errors, если ваш опыт обработки ошибок на уровне строки с помощью операторов On Error Resume Next и If Err Then недостаточен.

15. Теперь попробуйте запустить свое приложение. Вероятно, появятся некоторые ошибки выполнения, но они обычно легко исправляются в Debug Window (Окне Отладки).

Совет 73. Используйте параметр Optional при вызове процедур

В VB4 как процедуры-функции, так и подпрограммы могут использовать ключевое слово Optional (Необязательный) в объявлении процедур для указания необязательного параметра. Например:

```
Function mfbCheckDBStatus(Optional vroTest As Variant) As Boolean
```

В этом случае функция может вызываться без указания параметра:

```
MfbCheckDBStatus
```

Если необходимо проверить наличие необязательного параметра, используйте функцию IsMissing для его тестирования.

Совет 74. Как узнать разрешающую способность монитора

Для этого можно воспользоваться функцией WinAPI:

```
Declare Function GetsystemMetrics Lib "User" (ByVal _
  iIndex As Integer) As Integer

Sub Form_Load()
  Dim iXres As Integer, iYres As Integer
  ' разрешающая способность экрана монитора:
  iXres = GetsystemMetrics(0)  ' по оси X
  iYres = GetsystemMetrics(1)  ' по оси Y
```

```
End Sub
```

Совет 75. Анализ имени файла

Довольно часто бывает необходимо по полному имени файла определить имя его каталога или/и само имя файла внутри каталога. Для этого можно воспользоваться следующими процедурами (они будут работать в любой версии MS Basic, здесь используется формат представления текста модуля для QB):

```
DECLARE FUNCTION InstrReverse% (Text$, Key$)
DEFINT I-N

SUB FileNameTest (PathFile$, Path$, File$)
  '
  ' Вход:  PathFile$ - полное имя файла
  ' Выход: Path$ - имя каталога
  ' File$ - имя файла
  ' Например:
  ' Вход:  PathFile$ = "d:\Test1\Test2\File.txt"
  ' Выход: Path$ = "d:\Test1\Test2\"
  ' File$ = "File.txt"
  ''''''''''''''''''''''''''''''''''''''''''''
  in = InstrReverse(PathFile$, "\")
  IF in <= 0 THEN
    Path$ = "": File$ = PathFile$
  ELSE
    Path$ = LEFT$(PathFile$, in): File$ = MID$(PathFile$, in + 1)
  END IF
END SUB

FUNCTION InstrReverse (Text$, Key$)
  '
  '  поиск ПОСЛЕДНЕГО (а не первого, как в INSTR)
  '  контекста в строке
  '  Text$ - исходная строка
  '  Key$  - разделитель в строке
  ' Например:
  ' Вход:  Text$ = "d:\Test1\Test2\Test3"
  ' Key$ = "\"
  ' Выход: InstrReverse = 15
  ''''''''''''''''''''''''''''''''''''''''''''
  in = INSTR(Text$, Key$)
  IF in > 0 THEN
    DO WHILE in < LEN(Text$)
      in1 = INSTR(in + 1, Text$, Key$)
      IF in1 <= 0 THEN EXIT DO ELSE in = in1
    LOOP
  END IF
  InstrReverse = in
END FUNCTION
```

Андрей Колесов, Ольга Павлова

© 1997, Андрей Колесов, Ольга Павлова
Авторский вариант. Статья была опубликована с незначительной литературной правкой в журнале "КомпьютерПресс" N 1/97, с. 58-59.

Совет 76 . Как зарегистрировать OCX-элементы и OLE-сервера

При работе с VB3 никаких особых проблем с установкой на компьютер модулей VBX не было: модули можно было просто скопировать на дискету, затем переписать на жесткий диск и сразу начать с ними работать. Для элементов управления OCX такой метод не подходит — их надо не просто переписать, но и зарегистрировать. Аналогичная проблема возникает и с in-process OLE-серверами, которые можно разрабатывать в VB4.

Для регистрации модулей OCX и OLE-серверов необходимо воспользоваться утилитами REGOCX16.EXE и REGOCX32.EXE, которые можно найти в каталоге /TOOLS/PSS на компакт-диске с VB4. Первая из этих утилит предназначена для 16-разрядных элементов управления OLE, а вторая — для 32-разрядных. Будет полезно скопировать обе утилиты на свой жесткий диск. Объяснение процедуры регистрации приведем на примере элемента управления SYSINFO.OCX, находящегося на том же на компакт-диске.

Сначала скопируем OCX-файл на жесткий диск (обычно это делается в каталог WINDOWS\SYSTEM). Затем воспользуемся утилитой REGOCX32.EXE для регистрации этого элемента управления:

```
REGOCX32.EXE SYSINFO.OCX
```

После того как утилита завершит свою работу, в диалоговом окне References (оно вызывается из меню Tools) появится новый элемент, который теперь можно пометить как доступный. Наконец, в меню Tools выберите позицию Custom Controls и установите флажок в строке "Microsoft SysInfo Control".

Для регистрации in-process OLE-сервера потребуется использовать файл REGSVR32.EXE, который также можно найти в каталоге /TOOLS/PSS на компакт-диске с VB. Данная утилита поддерживает следующие опции:

1. регистрация in-process DLL-библиотеки:
2. `REGSVR32.EXE C:\MYDIR\MYINPROC.DLL`
3. отмена регистрации in-process DLL-библиотеки:
4. `REGSVR32.EXE /U C:\MYDIR\MYINPROC.DLL`

Эти утилиты можно использовать для регистрации элементов управления OCX, а также in-process DLL-библиотек, написанных в VB4. При создании автономных OLE-серверов они будут скомпилированы в исполняемый файл. Если дважды щелкнуть по OLE-серверу из Windows Explorer в Windows 95 или из Диспетчера Файлов (File Manager) в Windows 3.11, автономные OLE-серверы зарегистрируются самостоятельно. С другой стороны, можно зарегистрировать или отменить регистрацию автономных OLE-серверов из командной строки:

1. регистрация автономного сервера:
2. `C:>MYOLESVR.EXE /REGSERVER`

3. отмена регистрации автономного сервера:
4. C:>MYOLESVR.EXE /UNREGSERVER

Совет 77. Совместное использование массивов для нескольких форм

Проблема здесь заключается в том, что массив, зарезервированный в некоторой форме, нельзя описать как Public. Попытка такого объявления вызовет следующее сообщение об ошибке: "Constants, fixed-length strings, arrays, and Declare statements not allowed as Public members of class or form module". То есть для того чтобы сделать массив доступным, например в разных формах, его нужно поместить в общий программный модуль. Однако иногда бывает удобнее все же резервировать массив в модуле формы. Для доступа к нему из других форм можно предложить различные варианты.

Вариант 1. Следует объявить переменную Public Variant в форме, содержащей массив, а затем присвоить ей содержимое всего массива:

```
Option Explicit
Public Array As Variant
Private sArray (1 To 2) As String

Private Sub Form_Load ()
  sArray (1) = "Hello"
  sArray (2) = "World"
  Array = sArray ()
End Sub
```

Тогда другие формы смогут ссылаться на переменную Array, как если бы она была самим массивом:

```
Debug.Print frmOne.Array (1)
' Печатается 'Hello'
```

Вариант 2. Внутри формы можно создать процедуру Property Get, которая будет возвращать заданный элемент массива:

```
Public Property Get Array (ByVal iElement As Integer) As String
    Array = sArray (iElement)
End Property
```

Используйте ту же самую команду, чтобы сослаться на свойство из внешней формы:

```
Debug.Print frmOne.Array (1)
' Печатается 'Hello'
```

Совет 78. Как создать быстрые клавиши в Win95

Иногда бывает полезно уметь создавать быстрые клавиши в среде Windows 95 из программ, написанных на VB. Сделать это можно следующим образом.

В исходном коде, записанном в Setup Kit из VB4 (SETUP1.VBP), можно обнаружить следующее объявление функции:

Declare Function fCreateShellLink Lib "STKIT432.DLL" _ (ByVal lpstrFolderName As String _ ByVal lpstrLinkName As String _ ByVal lpstrLinkPath As String _ ByVal lpstrLinkArgs As String) As Long

Программа Setup обращается к этой функции, чтобы добавить быструю клавишу (в технической документации Microsoft она называется "Shell Link") для инсталлированной программы в меню Start в Win95. Не сразу очевидно, что данная функция может использоваться для создания быстрой клавиши в любой позиции каталога, содержащего меню Start.

Первый параметр этой функции (lpstrFolderName) относится к папке Programs из меню Start, которая в большинстве систем находится в каталоге \Windows\Start_Menu\Programs. Это означает, что если передать пустую символьную переменную в качестве первого параметра, Windows создаст быструю клавишу в самой папке Programs. Аналогично можно передвигаться от папки Programs к любому другому каталогу на жестком диске. Например, следующий код создает быструю клавишу на рабочем столе пользователя:

```
lResult = fCreateShellLink ("..\..\Desktop", _
  "Название быстрой клавиши", "d:\path\appname.exe", "")
```

Андрей Колесов, Ольга Павлова

© 1997, Андрей Колесов, Ольга Павлова
Авторский вариант. Статья была опубликована с незначительной литературной правкой в журнале "КомпьютерПресс" N 2/97, с. 74-81.

Совет 79. Как обойти все элементы дерева (TreeView Control)

(совет прислал Николай Баранов)

Для графического представления объектов, имеющих иерархическую структуру, очень удобно использовать элемент управления TreeView.OCX. При работе с ним часто возникает необходимость обхода всех элементов дерева или какой-то его ветви (например, для печати дерева на принтере).

Следующая рекурсивная процедура позволяет это легко выполнить:

```
Sub SeeTree(Level%)
  Dim nc As Node
  If Level% = 0 And (Not CurrentNode Is Nothing) Then
    Set CurrentNode = CurrentNode.FirstSibling
    ' вставьте сюда что-нибудь,
    ' что вы хотите сделать с элементом дерева
    ' например, MsgBox CurrentNode.Text,
    ' чтобы просто посмотреть
  End If
  While (Not CurrentNode Is Nothing)
    ' у элемента есть дети?
    If CurrentNode.Children > 0 Then
      Set CurrentNode = CurrentNode.Child
      ' сюда тоже можно что-нибудь вставить
      Level% = Level% + 1
      SeeTree Level   ' рекурсия здесь
```

```
    End If
    ' следующий элемент на этом уровне
    Set nc = CurrentNode.Next
    ' элемент есть - идем дальше по уровню
    If Not nc Is Nothing Then
      Set CurrentNode = nc
      ' вставьте сюда что-нибудь,
      ' что вы хотите сделать с элементом дерева
      ' например, MsgBox CurrentNode.Text,
      ' чтобы просто посмотреть
    Else ' элемента нет - идем на верхний уровень
      ' следующая строка пригодится,
      ' если есть желание развернуть все на экране
      ' CurrentNode.EnsureVisible
      '
      Set CurrentNode = CurrentNode.Parent
      Level% = Level% - 1
      Exit Sub
    End If
  Wend
End Sub
```

В секцию Declarations формы, в которой расположен TreeView, нужно добавить такую строку:

```
Dim CurrentNode As Node
```

Вызов процедуры может выглядеть примерно так:

```
' для обхода всех элементов дерева,
' находящихся на одном уровне с выделенным:
Set CurrentNode = TreeView1.SelectedItem
...
' для обхода всех элементов-детей
' выделенного элемента дерева
Set CurrentNode = TreeView1.SelectedItem.Child
...
' переменная, показывающая текущий уровень иерархии
Level% = 0
SeeTree Level%
```

Совет 80. Будьте внимательны при работе с константами типа Long

(Важное дополнение к этому совету содержится в Совете149[4].)

Николай Баранов, приславший предыдущий совет, сообщил также, что заметил ошибочную ситуацию при использовании логических операций при работе с переменными типа Long(&). Проблема выглядит следующим образом.

Например, при работе с OLE-Automation серверами приходится анализировать коды возвращаемых ошибок, которые представляют собой длинное целое, например, что-нибудь вроде -2147221229 (&H80040113 - MAPI_E_USER_CANCEL). Однако вся необходимая информация об ошибке содержится в младших двух байтах, и вопрос лишь в том, как их достать. Но это оказывается совсем не так просто.

Например, если выполнить такой фрагмент программы:

[4] http://www.visual.2000.ru/develop/ms-vb/tips/9812.htm

```
Code& = &H80040113
Result& = Code& And &HFFFF
```

то содержимое Result& будет равно не &H0113, как хотелось бы, а &H80040113. Ситуация не изменится, даже если написать по-другому:

```
Const Mask& = &HFFFF
Code& = &H80040113
Result& = Code& And Mask&
```

Более того, неверно работают и другие логические операции. Например, в результате выполнения

```
Result& = Code& Or Mask&
```

получится не &H8004FFFF, а &HFFFFFFFF.

Такие ошибки (или странности) происходят только с использованием масок, занимающих не более 16 младших разрядов, для значений же больше &HFFFF все работает нормально. В связи с этим для выделения двух младших байтов Николай предлагает использовать, например, такую конструкцию:

```
(&H80040113 And &H7FFFFFFF) And &H8000FFFF
```

Действительно, ситуация выглядит довольно странно и неприятно. Ведь, если рассматривать общий случай логических операций двух переменных m1& и m2&, содержимое которых заранее неизвестно, то положение представляется почти безысходным (по крайней мере, универсальное решение проблемы будет весьма громоздким).

Однако при более детальном рассмотрении все оказывается не так уж и страшно. На самом деле никаких ошибок в логических операциях с переменным Long нет, а есть проблема формирования констант, которая связана с преобразованием данных из Integer в Long и наоборот. В связи с этим нужно вспомнить, что целочисленные переменные являются переменными со ЗНАКОМ, и нужно быть очень внимательным при переходе от беззнакового представления числа (&H...) к знаковому (цифровому десятичному). Рассмотрим такую конструкцию:

```
Mask& = &HFFFF
Print HEX$(Mask&)
' будет напечатано &HFFFFFFFF !!!!
```

Дело в том, что константа &HFFFF автоматически представляется в виде переменной Integer (попадает в диапазон данных) и в числовом выражении равна -1. Соответственно, при присвоении Mask& = &HFFFF происходит преобразование из Integer в Long и переменная Mask& = -1 (&HFFFFFFFF)! Аналогичные преобразования происходят и в приведенной выше операции Result& = Code& And &HFFFF.

В этой ситуации в вину разработчикам VB можно поставить только двусмысленность операции определения константы в явном виде:

```
Const Mask& = &HFFFF
Print HEX$(Mask&)
' будет напечатано &HFFFFFFFF !!!!
```

Здесь неожиданно константа &HFFFF опять интерпретируется как Integer, хотя она обозначена как Long. Это тоже не очень хорошо, но вполне разрешимо - надо только помнить об этом свойстве констант типа Long. Все будет работать так, как надо, если использовать следующие варианты: Const Mask& = 65635 или Mask& = &H10000 - 1.

Но конструкция Const Mask& = &H10000 - 1 опять приведет к нежелательному результату (&HFFFFFFFF).

В результате можно сделать следующий вывод: для определения констант типа Long или их использования в арифметических или логических операциях в диапазоне значений &H8000-&HFFFF (32768-65535) нельзя использовать беззнаковое шестнадцатеричное представление, а можно применять только десятичное.

Совет 81. Как обеспечить числовой ввод

Часто бывает необходимо ограничить ввод данных в текстовое окно только числовым значением. VB содержит элемент управления для замаскированного редактирования, однако пользоваться им в некоторых случаях не очень удобно. Вместо этого можно применять следующую функцию для проверки числового ввода в VB4. Для работы с ней в VB3 измените два логических параметра на целочисленные:

```
Function ValidNumber (iAscii As Integer, txtBox As TextBox, _
  bSign As Boolean, bPoint As Boolean) As Integer
  ' Ввод символа по умолчанию
  ValidNumber = iAscii
  Select Case iAscii
    Case 8  ' Клавиша Backspace
    Case 43, 45
      ' Ввод символа знака (+, -) только в том случае,
      ' если флаг знака равен "Истина" (True), а сам
      ' символ стоит первым по порядку
      If (Not bSign) Or (txtBox.SelStart > 1) Then
        ValidNumber = 0
      End If
    Case 46
      ' Ввод десятичной точки только в том случае,
      ' если флаг равен "Истина" (True) и в строке нет
      ' ни одной другой точки
      If (Not bPoint) Or (InStr(txtBox.Text, ".")) Then
         ValidNumber = 0
      End If
    Case 48 To 57  ' Цифры от 0 до 9
    Case Else  ' Все остальное
      ValiNumber = 0
  End Select
End Function
```

Данная функция должна вызываться из события KeyPress текстового окна, например, так:

```
Private Sub txtNumber_KeyPress (KeyAscii As Integer)
  KeyAscii = ValidNumber (KeyAscii, txtNumber, True, True)
End Sub
```

Совет 82. Как выполнить доступ к защищенным базам данных из VB4

Даже прочитав документацию VB4 о том, как открыть защищенную базу данных Access, бывает совсем не просто разобраться в этом вопросе. Одна из причин - нечеткое описание

того, какие команды относятся к 16-разрядными программам, а какие - к 32-разрядным. Кроме того, очевидный технический дефект в 16-разрядной версии для баз данных Jet 2.5 приводит к тому, что метод CreateWorkspace из DBEngine не работает до тех пор, пока свойствам DefaultUser и DefaultPassword не будут присвоены значения, соответствующие защищенной базе данных.

Здесь приводится пример операций, необходимых для открытия защищенной базы данных как в 16-, так и в 32-разрядных приложениях. Для 16-разрядных программ сначала создайте файл APPNAME.INI (где APPNAME - это имя исполняемого файла вашей программы), который содержит по крайней мере следующее:

```
[Data]
Database = D:\PATH\DBNAME.MDB
[Options]
SystemDB = D:\PATH\SYSTEM.MDA
```

Затем, чтобы открыть базу данных, выполните следующее:

```
Dim sUserName As String
Dim sPassword As String
Dim db As Database
Dim ws As Workspace
sUserName = "Здесь находится ваше имя"
sPassword = "Здесь находится ваш пароль"
' Создание защищенного рабочего пространства
With DBEngine
  .IniPath = "D:\PATH\APPNAME.INI"
  .DefaultUser = sUserName
  .DefaultPassword = sPassword
End With
' Имя рабочего пространства является
' произвольным, но должно быть уникальным
Set ws = DBEngine.CreateWorkspace ("Name", _
sUserName, sPassword)
' Открытие базы данных через защищенное
' рабочее пространство
Set db = ws.OpenDatabase (D:\PATH\DBNAME.MDB"...)
```

32-разрядные программы не требуют файла INI, кроме того, не нужно определять свойства DefaulUser и DefaulPassword. Установите свойство SystemDB из DBEngine таким образом, чтобы оно указывало на вашу системную базу данных:

```
Dim sUserName As String
Dim sPassword As String
Dim db As Database
Dim ws As Workspace
  sUserName = "Здесь находится ваше имя"
  sPassword = "Здесь находится ваш пароль"
  ' Создание защищенного рабочего пространства
  DBEngine.SystemDB  = "D:\PATH\SYSTEM.MDW"
  Set ws = DBEngine.CreateWorkspace ("Name", _
    sUserName, sPassword)
  ' И это все! Сезам, откройся ...
  Set db = ws.OpenDatabase ("D:\PATH\DBNAME.MDB"...)
```

Совет 83. Используйте некоторые правила написания кода в VB3 для его простого переноса в VB4

Вполне возможно, что по каким-то причинам вы продолжаете работать в VB3, понимая, что в будущем все же придется переводить свои программы в VB4. Поэтому имеет смысл использовать несколько правил написания кода в VB3, которые намного упростят такой переход.

Строки, содержащие несколько операторов

VB4 обрабатывает такие строки не так, как VB3. В VB4 первый оператор в строке воспринимается как метка, если он состоит из одного ключевого слова. Самый простой пример:

```
Beep: Beep: Beep
```

В VB3 вы услышите три гудка, в VB4 - два. Так что, если вы любите объединять несколько строк в одну, пора распрощаться с этой привычкой.

Конкатенация строк

Хотя программисты в течение долгого времени предпочитали объединять строки при помощи знака "амперсанд" (&), поскольку это более быстрый и надежный способ, тем не менее у них также была возможность использовать знак "плюс" (+). Однако время, когда у программистов был выбор, прошло. В VB4 необходимо использовать только знак "амперсанд", так как оператор "плюс" в некоторых случаях имеет иное действие при работе со строками.

Ограничение в 64К

Ограничение в 64К на размер исходного кода в одной процедуре не изменилось со времен VB3. Тем не менее в самом языке произошли некоторые изменения, так что есть смысл еще раз рассмотреть это ограничение.

Поскольку VB4 использует в своей языковой основе более общий стандарт VBA, он создает код, который не столь оптимизирован, как в VB3. И хотя полученный в VB4 код предназначен для более широкого диапазона процессоров, он слегка увеличивает размеры процедуры. Поэтому, если размеры исходного кода в какой-либо из процедур приближаются к 64К, необходимо разбить ее на несколько более мелких процедур.

Совет 84. Как включить 16-разрядную версию Crystal OCX в прикладную программу

Такая проблема возникает при работе в 16-разрядной версии VB4, в которой вы создаете некое приложение, использующее 16-разрядную версию Crystal OCX. Если это приложение предназначено для дальнейшего распространения (продажи), то можно создать его дистрибутив с помощью средства Microsoft VB Distribution Expert. Однако когда пользователи пытаются установить ваше приложение на свой компьютер, то они получают сообщение "Missing CRXLAT16.DLL" ("Не найдена библиотека CRXLAT16.DLL"). То есть программа установки ищет библиотеку CRXLAT16.DLL, чтобы присоединить ее к вашему приложению, но не находит такого файла.

Данная ситуация - ошибка, для исправления которой необходимо отредактировать файл SWDEPEND.INI. Найдите в нем ссылку на библиотеку CRXLAT16.DLL и замените ее на CRXLATE.DLL. Именно такой совет был опубликован в разделе Tech Tips ("Технические советы") в летнем выпуске Crystal Reporter - бесплатно распространяемом издании, которое выпускается компанией Seagate Software (бывшей Crystal Services).

Совет 85. Как сохранить текущее значение индекса списка

Ситуация может выглядеть таким образом. При работе со списком вы выделяете некую строку, затем переходите в другой элемент управления на форме и возвращаетесь обратно в список, щелкнув его мышью. Но при этом вы хотите, чтобы в качестве выделенного элемента списка оказался не попавший под курсор мыши, а установленный как текущий при предыдущем выходе из окна.

Решение здесь очень простое: в момент выхода из списка в процедуре LostFocus нужно запомнить состояние свойства ListIndex, а в момент возврата - восстановить это состояние в процедуре GotFocus. В принципе, для этого можно использовать какую- нибудь переменную, глобальную на уровне данной формы, но сохранение значения индекса в свойстве Tag, имеющего тип строковой переменной и применяемого специально для хранения любых данных для дальнейшего их использования, выглядит более элегантно.

Сохраните свойство ListIndex в тот момент, когда окно списка теряет фокус:

```
Sub List1_LostFocus()
  List1.Tag = Format$(List1.ListIndex)
End Sub
```

Соответственно при возврате в список восстановите значение индекса:

```
Sub List1_GotFocus()
  If List1.Tag <> "" Then List1.ListIndex = Cint(List1.Tag)
End Sub
```

Обратите внимание, что, если в процедуре GotFocus убрать проверку значения Tag, то при первом входе в список произойдет ошибка, так как свойство Tag не было еще определено.

Совет 86. Как уменьшить размеры приложения

Здесь приводятся некоторые обобщенные рекомендации экспертов журнала VBPJ (9'96) по этому поводу:

Исходный код:

- Поместите общий код в подпрограммы и процедуры-функции.
- Используйте условную компиляцию для того, чтобы избавиться от кода, применяемого только для отладки и различных операционных систем.
- Удалите ненужный код и неиспользуемые операторы Const и Declare (каждый оператор Declare требует 40 байт); применяйте API Viewer для добавления объявлений процедур и Code Profiler для обнаружения никогда не выполняемых процедур.
- Используйте параметры Optional для уменьшения количества параметров, передаваемых в процедуры.
- VB загружает модули только по требованию, поэтому поместите все связанные функции в один модуль.

- Используете блоки With...End With и промежуточные объекты для уменьшения количества точек при ссылке на вложенный объект.
- Избегайте использования рекурсивных процедур.
- Используйте функции Choose и Switch вместо Select Case и If...ElseIf.
- Используйте встроенные константы vbTab, vbNullChar и vbCrLf вместо функций Chr() (это также немного ускоряет процесс выполнения приложения).
- Перед тем как приступить к написанию процедуры или обратиться к внешнему компоненту, поищите эквивалентную API-функцию. Эти функции можно использовать для копирования файлов, сравнения записей, поиска строк в окне списка и множества других функций - как общих, так и специальных.

Переменные

- Используйте локальные, а не глобальные переменные и объекты (это также улучшает повторную исполняемость модуля).
- Устанавливайте объекты равными Nothing ("Пусто"), как только в них отпадает необходимость.
- Помните, что структуры Type могут встраивать как строки переменной длины, так и динамические массивы.
- Лучше используйте динамические, а не статические массивы. Создавайте небольшие массивы, а затем увеличивайте их размеры с помощью оператора ReDim Preserve только в тот момент и в том случае, если это необходимо.
- Храните таблицы сообщений в файлах (или в файлах ресурсов) и загружайте их только по мере надобности.

Формы и элементы управления

- Всегда выгружайте формы, а не просто прячьте их.
- Устанавливайте формы равными Nothing ("Пусто"), как только выгрузите их (например, Set Form1 = Nothing), чтобы освободить память, используемую переменными формы.
- Загружайте окна списка или поля ввода со списком в процессе разработки, а не во время выполнения.
- Сохраняйте код, группируя связанные позиции меню и элементы управления (такие как кнопки-переключатели) в массивах элементов управления.
- Используйте массивы элементов управления для загрузки дополнительных элементов управления во время выполнения приложения; не определяйте их в процессе разработки.
- Копируйте атрибуты шрифта между консолями, используя свойство new Font. Не копируйте каждое конкретное свойство Fontxxxx.
- Используйте метод Move ("Передвинуть") вместо того, чтобы присваивать значения свойствам Left, Top, Width и Height.

Средство JET

- Поскольку мгновенные "снимки" (snapshots) включают все выделенные поля, а не просто ключевые значения, используйте dynasets, если можете.
- Используйте SQL для получения только тех записей и полей, которые вам действительно необходимы.
- Используйте хранящиеся процедуры (объекты QueryDef), а не динамические SQLзапросы, чтобы избежать загрузки компилятора запросов во время выполнения приложения. Ускорьте компиляцию запроса путем выполнения этого запроса по крайней мере один раз перед созданием EXE файла.
- Используйте метод Clone вместо создания нового набора записей тогда, когда вам это удобно.

- Если вы используете транзакции для ускорения обработки базы данных, задайте промежуточные команды Commit для переноса на диск информации, хранящейся во внутреннем буфере.
- Не используйте средство Jet для небольших наборов данных, которые могут легко храниться в массиве записей, - это сэкономит размеры оперативной памяти и дисковое пространство.

Совет 87. Будьте внимательны при обращениях к функциям API

Мы уже неоднократно говорили, что основной причиной ошибок при обращении к функциям API является неверная передача параметров (их числа, типов данных, адресации переменных и пр.).

Вот характерный пример, присланный нашим читателем:

```
'========== В модуле BAS ================
Declare Function Polygon Lib "GDI" (ByVal hDC _
  As Integer, lpPoints As POINTAPI, ByVal nCount _
  As Integer) As Integer
  Type POINTAPI
    x As Integer
    y As Integer
  End Type
  Dim m1(1 to 4) as POINTAPI

'========== В форме ====================
' точки  функции, которая будет
' обрабатываться функцией Polygon
  m1(1).x = 130: m1(1).y = 40
  m1(2).x = 90: m1(2).y = 60
  m1(3).x = 80: m1(3).y = 90
  m1(4).x = 30: m1(4).y = 40
  fz% = Polygon(Form1.hDC, m1(1), 4)
```

Вопрос: *Почему в функции Polygon() во втором параметре надо указывать первый элемент массива, а не сам массив, как это принято при обращении к функциям VB?*

Наш ответ таков:

В Basic массивы и строковые переменные имеют специальные описатели из нескольких байт. Например, в описателе массива хранятся адрес области самих данных, размерность массива, верхняя и нижняя граница каждого массива.

При передаче данных между процедурами в виде m1() или Sym$ на самом деле передается адрес этого описателя, которой умеют понимать только процедуры, написанные на Basic. Универсальным же способом передачи данных, используемым во многих других языках программирования (C, Fortran, Pascal), является передача адреса памяти данных. При этом предполагается, что вызываемая подпрограмма должна сама знать о том, каков тип этих данных, их структура, число индексов (для массива) и пр. Если какие-то из этих характеристик являются переменными, то их нужно передавать в виде дополнительных параметров.

Именно такие универсальные способы передачи параметров используются при обращении к функциям API. Обратите внимание, что в обращении fz% = Polygon(Form1.hDC, m1(1), 4) во втором параметре передается адрес начала массива данных, а в третьем - число его

элементов. При этом важно отметить, что вся ответственность за правильное описание структуры массива в вызывающей программе полностью ложится на программиста. Если вы в Type POINTAPI, например, укажите еще одно поле z, то никакой диагностики выдано не будет, но функция Polygon правильно работать уже не будет.

Совет 87a. Как проиграть Wav-файл

В конце 1996 года мы получили почти одновременно такие два письма читателей:

*"Я долгое время работаю на VB и нигде не могу узнать,- как работать со звуковыми файлами *.WAV кроме как исполь- зовать SOUND RECORDER. Если можете - ПОМОГИТЕ !!!"*

"У меня возник такой вопрос : как на VB 4.0 при нажатии кнопки мыши проиграть определснный WAV файл."

Мы не знали ответа (не работали со звуком) и хотели опубликовать вопросы в журнале с подзаголовком "Вопросы без ответов". Однако наш редактор журнала "КомпьютерПресс" быстро написал нужное решение и опубликовал его с подзаголовком **"Вопросов без ответов не бывает!"**.

Ответ Алексея Федорова:

Чтобы по нажатию кнопки мыши воспроизводился WAV-файл, Чтобы проиграть WAV-файл следует воспользоваться стандартной функцией Windows API sndPlaySound, которая находится в системной DLL-библиотеке WINMM.DLL. Эта функция имеет два параметра: первый указывает имя WAV-файла, второй является флагом, определяющем вид выполняемой функции:

```
Private Declare Function sndPlaySound _
  Lib "winmm.dll" Alias "sndPlaySoundA" _
  (ByVal lpszName As String, ByVal dwFlags As Long) As Long
'
' Описание значений параметра dwFlags:

Const SND_SYNC = &H0      ' Файл воспроизводится синхронно
      ' и функция не возвращает управление до окончания воспроизведения

Const SND_ASYNC = &H1  ' Файл воспроизводится асинхронно
      ' и функция возвращает управление сразу же после
      ' начала воспроизведения. Для того, чтобы
      ' прервать воспроизведение, необходимо вызвать
      ' функцию sndPlaySound с именем файла, равным ""

Const SND_NODEFAULT = &H2 ' Указывает на то, что если файл,
      ' заданный первым параметром, не найден, то не должен
      ' воспроизводиться файл по умолчанию

Const SND_MEMORY = &H4 ' Указывает на то, что имя файла
      ' соответствует WAV-файлу, находящемуся в памяти,
      ' например, загруженному из ресурса

Const SND_LOOP = &H8  ' Файл воспроизводится от начала до
      ' конца бесконечное число раз до тех пор, пока не
      ' вызвана функцию sndPlaySound с именем файла, равным "".
      ' При таком воспроизведении должен быть указан и флаг
      ' SND_ASYNC
```

```
Const SND_NOSTOP = &H10    ' Функция возвращает FALSE, если
      ' в момент ее вызова уже воспроизводится какой-нибудь файл

' Соответственно, реальное обращение
' к функции sndPlaySound может выглядеть примерно так

 Result = sndPlaySound("c:\wav\demo.wav", SDN_ASYNC)
```

Теперь еще один вопрос — как сделать, чтобы файл проигрывался при щелчке мыши по форме? Для этого нужно поместить обращение к sndPlaySound в событийную процедуру:

```
Sub Form_MouseDown(Button As Integer, _
    Shift As Integer, X As Single, Y As Single)
'
  Result = sndPlaySound("c:\wav\demo.wav", SDN_ASYNC)
End Sub
```

В этом случае файл будет воспроизводится при нажатии кнопки мыши в любой части клиентской области окна. Чтобы разделить эту область на подобласти, для каждой из которых воспроизводится свой WAV-файл, следует проверять координаты, в которых произошло нажатие кнопки мыши. Они передаются в качестве параметров обработчика MouseDown. Если же проверка координат не требуется, то можно использовать обработчик события Click.

И еще одно замечание, если области на экране достаточно большие, то можно использовать кнопки, сделав их невидимыми. В обработчике нажатия каждой кнопки на соответствующем элементе управления с помощью функции sndPlaySound будет воспроизводить необходимый WAV-файл.

Андрей Колесов, Ольга Павлова

© 1997, Андрей Колесов, Ольга Павлова
Авторский вариант. Статья была опубликована с незначительной литературной правкой в журнале "КомпьютерПресс" N 3/97, с.116-120.

Совет 88. Будьте внимательны при установке VB 5.0.

У вас могут появиться некоторые проблемы при установке VB 5.0 (в том числе и свободно распространяемой редакции VB5/CCE), если на компьютере уже стоит VB 4.0. На это особенно стоит обратить внимание тем, кто рискует работать с бета- версиями, уже вовсю распространяющимися на "китайских" CD.

Дело в том, что VB 5.0 и VB 4.0 используют ряд одних и тех же элементов управления OCX. Соответственно, при установке VB 5.0 он автоматически заменяет эти модули на свои. С чисто юридической точки зрения может возникнуть проблема с дистрибьюцией приложений, написанных для VB 4.0, в которые могут попасть компоненты VB 5.0. Об этом были специально предупреждены бета-тестеры. Но могут возникнуть трудности и чисто технического характера. Поясним это на собственном примере.

Мы установили на свой ПК, где уже был VB 4.0, сначала VB5/CCE (бета-версия 1), а потом VB5/Pro (официальная бета-версия 2). Неожиданно при запуске утилиты API Text

Viewer (VB 4.0) появилось сообщение об ошибке: "Модуль COMDLG32.OCX отсутствует или неверно лицензирован". Потом обнаружилось, что и другая утилита не хочет работать с COMCTL32.OCX. Однако сам VB 4.0 работал с этими модулями нормально. После небольшого анализа прояснилась следующая ситуация.

Оказалось, что на CD находилось по две версии этих модулей: для VB5/CCE с датой 25.10.96 и для VB5/Pro с датой 01.10.96. Соответственно при их установке автоматически остались модули с более поздней датой (VB5/CCE). Мы подумали, не возникли ли проблемы из-за каких-то наших ошибок при установке (мы проводили установку в несколько приемов, что-то добавляя, что-то удаляя и пр.), и решили заново провести полную инсталляцию. Однако после удаления VB5/Pro с компьютера исчезли все его OCX: он посчитал их своими собственными компонентами! Остались лишь файлы COMDLG32.OCX и COMCTL32.OCX, которые он, по-видимому, определил как чужие из-за их даты. Короче говоря, VB 4.0 оказался без расширенного набора элементов управления.

Повторив установку VB5/Pro, мы обнаружили те же самые ошибки. Тогда, заподозрив дефект именно в модулях от 25.10.96, мы опять удалили VB5/Pro (для чистоты эксперимента), переименовали на жестком диске файлы COMDLG32.OCX и COMCTL32.OCX от 25.10.96 и повторили установку. С версиями модулей от 01.10.96 все вроде бы стало работать нормально.

Хотелось бы подчеркнуть, что речь здесь шла не о конечных версиях продукта, а об их бета-версиях. Однако описанная ситуация является довольно типичной для проблемы обновления версий вообще. В связи с этим нужно сказать, что идея компонентной организации вычислительных систем (когда разные приложения используют одни и те же компоненты — OCX, DLL и пр.) является безусловной прогрессивной и заманчивой, но в процессе ее практической реализации возникают многие вопросы, в том числе касающиеся обновления версий компонентов, контроля их принадлежности (например, когда при удалении приложения оно заодно удаляет и общие модули) и т.п. К тому же, несмотря на все желания разработчика, не всегда удается добиться полной совместимости "снизу вверх". Или у вас вдруг могут возникнуть проблемы с оперативной памятью из-за увеличения размеров новых версий компонентов. Так что будьте внимательны.

Совет 88a. Индексирование русскоязычных баз данных

Наш читатель обратился к нам с такой проблемой:

При создании базы данных в VB 4.0 с помощью Data Manager для русскоязычных строковых переменных не создается индекс и не работают методы поиска и сортировки. Однако если создать БД с помощью русскоязычного ACCESS в русскоязычной версии Windows, то потом с ней можно работать в VB без всяких проблем. Создавать БД в среде VB без Data Manager я не пробовал, может быть, там этих проблем и нет.

Ответ

При работе в VB правила сортировки и поиска в базах данных в самой программе устанавливаются с помощью параметра locale при создании и преобразовании БД (методы CreateDatabase и CompactDatabase). После создания базы данных значение параметра фиксируется (для объектов DataBase и Field) в виде свойства CollatingOrder, которое можно прочитать (но не изменить!) в ходе работы программы:

```
object.CollatingOrder
```

Этот параметр можно использовать в программе для настройки операции сравнения и поиска строковых переменных, создания новых объектов (например, баз данных) и пр.

Значения наиболее актуальных вариантов параметра locale для VB 4.0 (в нем предусмотрено 20 значений для разных языков) таковы:

```
dbSortGeneral = 1033 = &409     ' English
dbSortCyrillic = 1049 = &419    ' Russian - в VB 3.0 его нет
dbSortNeutral = 1024 = &400     ' без сортировки
```

(Для VB 3.0 состав кодов меньше, русского языка там нет, символьные названия констант и их значения — другие).

Эти параметры также можно задавать и при работе в Data Manager. Создавая и открывая базу данных, выделите в списке нужную таблицу и нажмите кнопку Design. В списке описателей полей Вы увидите параметр CollatingOrder, который задается в цифровом виде (конечно, было бы удобнее иметь дело со списком типа General, Cyrillic...). Затем выберите нужное поле, нажмите Edit и т.д.

Совет 89. Управляйте режимами преобразования, сортировки и сравнения символьных переменных.

Учитывая предыдущий вопрос читателя, имеет смысл подробнее рассмотреть некоторые аспекты работы с символьными переменными, в том числе с русскими буквами. Вообще говоря, переменная String представляет собой простую последовательность байтов. Соответственно операции сравнения (на них были основаны также алгоритмы поиска и сортировки) сводятся к последовательному сравнению значений байтов строки (кодов символов). Однако в частном (но наиболее часто встречающемся) случае строка является осмысленным набором символов, которые записаны в виде ASCII кода. А для них операции побайтового сравнения кодов не всегда подходят.

Например, для варианта русской кодовой таблицы ASCII последовательность сортировки (в порядке возрастания кодов) выглядит так:

```
ABCD...abcd...АБВГ...абвг
```

Для алфавитной же сортировки требуется другая последовательность (именно она реализуется при работе с базами данных):

```
aAbBcCdD...aАбБвВгГ
```

А для операций поиска необходимо часто вообще игнорировать регистр букв (прописные/строчные) (ignore case). В старых версиях Basic операторы Ucase/Lcase (преобразование всей строки в прописные или строчные буквы) являются фактически единственными командами, позволяющими производить какие-либо специфические операции над символьными переменными. Но они работали только с английскими буквами!

Впервые алгоритмы алфавитной сортировки и поиска были реализованы в базах данных ISAM, включенных в MS Basic PDS 7.1. В них была предусмотрена поддержка нескольких национальных языков, но русского среди них не было. В VB 3.0 в дополнение к работе с

БД (а, может быть, и раньше: с VB 2.0 мы не работали) появились аналогичные возможности на языковом уровне — операторы InStr и StrComp. В VB 4.0 была реализована также поддержка для русского языка.

Преобразование символьных данных

В VB 4.0 операторы Lcase/Ucase выполняют соответствующие преобразования не только над латинскими буквами (как в VB3), но и со всеми буквами установленной кодовой таблицы Windows 95. (У нас установлена 32-разрядная версия VB4. Может быть, это работает и для Windows 3.x? Поделитесь опытом.) Таким образом, для cp251 результатом функции Ucase$("Вася") будет: ВАСЯ. В VB 4.0 появилась также новая функция StrConv, которая выполняет дополнительные варианты преобразования кодов (в том числе аналог операций Ucase/Lcase в Unicode и обратно).

Операции сравнения

В VB есть два оператора: поиска — InStr (расширенный вариант оператора INSTR, используемого в Basic/DOS) и сравнения — StrComp. В них имеется возможность использования дополнительного (необязательного) параметра Compare:

- InStr ([start],string1,string2,[compare]) - поиск контекста;
- StrComp (string1,string2,[compare]) - сравнение двух переменных;
- Compare = 0 - двоичная обработка в соответствии со значениями кодов ASCII;
- Compare = 1 - алфавитная обработка в режиме ignore case (A=a, Б=б, Z= z и пр.) для текущей кодовой таблицы Windows 95;
- Compare = CollatingOrder (см. выше) алфавитная обработка в режиме ignore case для заданного национального языка.

В случае отсутствия параметра Compare в операторе используется его значение по умолчанию — 0 (двоичная обработка). Однако этот параметр по умолчанию можно изменить для всех процедур конкретного модуля с помощью оператора Option Compare Binary/Text: Binary указывает на режим двоичного сравнения (он установлен по умолчанию), Text — на алфавитный режим для текущей кодовой таблицы Windows.

Примеры (VB4, cp1251):

```
Print StrConv ("Петя петухов", 3)  ' = Петя Петухов
Print InStr ("Петя петухов", "п", 0)   ' = 6
Print InStr ("Петя петухов", "п", 1)   ' = 1
Print StrComp ("Петя", "петЯ", 1)  ' = -1
Print StrComp ("Петя", "петЯ", 0)  ' = 0
```

Совет 90. Простой алгоритм преобразования прописных букв в строчные и наоборот

ASCII-таблица латинских символов организована таким образом, что коды строчных букв (a-z = &H61-&H7A) отличаются от кодов соответствующих прописных (A-Z = &H41-&H5A) только наличием 1 в третьем разряде байта (&H20). Более того, данный принцип реализован для Windows-кодировки (cp1251) русских букв: а-я = &HE0-&HFF, А-Я = &HC0-&HDF.

Знание этой особенности кодировки может иногда пригодиться для каких-либо нетривиальных преобразований символьных данных, в том числе при программировании

на языках, не имеющих операторов типа Ucase/Lcase (например, на ассемблере). Вот как может выглядеть фрагмент такого преобразования, когда прописные буквы меняются на строчные, а строчные — на прописные:

```
Word$ = "aBcDeF"
For i% = 1 to Len(Word$)
  Mid$(Word$,i%) = Chr$(Asc(Mid$(Word$,i%)) Xor &H20)
Next
Print Word$   'напечатано - AbCdEf
```

Совет 91. Установка специальных шрифтов приложения

Бывают ситуации, когда в состав вашего приложения входят какие-то специальные шрифты, которые распространяются вместе с ним. Соответственно эти шрифты должны входить в состав дистрибутива и устанавливаться на жесткий диск в процессе инсталляции, то есть их надо уметь аккуратно добавлять к системе. Для этого можно использовать API-функцию LoadFontResource. Однако вначале следует самостоятельно скопировать файл в то место, где хранятся остальные шрифты. В Windows файл шрифта копируется в системный каталог, а в Windows 95 — в каталог \Windows\Fonts.

После вызова функции LoadFontResource нужно известить все открытые окна о том, что таблица шрифтов была изменена. Для этого отправьте в HWND_BROADCAST сообщение WM_FONTCHANGE, которое даст указание Windows послать данное сообщение во все открытые окна.

Процедура AddFont копирует файл шрифтов, регистрирует данный шрифт, а затем уведомляет об этом все открытые окна. Она представлена в варианте для VB 4.0. Для VB 3.0 в этом тексте нужно убрать фрагмент, относящийся к 32-разрядной версии, и объявление Private в операторах Declare:

```
Option Explicit
 #If Win32 Then
    Private Declare Function AddFontResource Lib _
            "gdi32" Alias "AddFontResourceA" (ByVal _
             lpFileName As String) As Long
    Private Declare Function SendMessage Lib _
             "user32" Alias "SendMessageA" (ByVal hWnd _
             As Long, ByVal wMsg As Long, ByVal wParam _
             As Long, lParam As Long) As Long
     Private Declare Function GetWindowsDirectory _
             Lib "kernel32" Alias "GetWindowsDirectoryA" _
             (ByVal lpBuffer As String, ByVal nSize As _
             Long) As Long
 #Else
     Private Declare Function AddFontResource Lib _
             "GDI" (ByVal lpFilename As Any) As Integer
     Private Declare Function SendMessage Lib "User" _
             (ByVal hWnd As Integer, ByVal wMsg As Integer, _
             ByVal wParam As Integer, lParam As Any) As Long
     Private Declare Function GetWindowsDirectory _
             Lib "Kernel" (ByVal lpBuffer As String, ByVal _
             nSize As Integer) As Integer
 #End If

Const WM_FONTCHANGE = &H1D
Const HWND_BROADCAST = &HFFFF&

Function AddFont3 (FileName As String) As Integer
```

```
  Dim WindowsDir As String
  Dim Lbuf As Long
  ' - Получение каталога Windows
  Lbuf = GetWindowsDirectory (WindowsDir, Len (Buffer))
  If Lbuf Then
    WindowsDir = Left$(windowsDir, Lbuf)
    ' - Копирование файла
    FileCopy FileName, WindowsDir & "\Fonts\" & FileName
    ' - Добавление шрифта
    If AddFontResource (FileName) Then
      SendMessage HWND_BROADCAST, WM_FONTCHANGE, 0, 0
      AddFont3 = True
    End If
  End If
End Function
```

Совет 92. Автоматическая установка кавычек в строковых переменных

Наличие внутри строковой переменной двойных или одинарных кавычек — довольно обычная проблема. Путаница же начинается тогда, когда такие переменные используются для составления сложных символьных выражений, не допускающих применения подобных символов: их появление внутри отдельного слова требует специального выделения самого слова с помощью все тех же кавычек. Пример таких символьных выражений — SQL-операторы.

Чтобы как-то автоматизировать операции расстановки кавычек, можно создать функцию, которая получает символьную переменную и проверяет, есть ли там двойные кавычки, внедренные в саму символьную переменную. Если они есть, тогда данная функция помещает одиночные кавычки в начало и конец переменной. В противном случае она помещает двойные кавычки в начало и конец переменной и возвращает результат вызывающей функции:

```
SQLx = "INSERT UserTable VALUES ("
SQLx = SQLx & Index & ","
SQLx = SQLx & q(txtUserName.text) & ","
SQLx = SQLx & q(txtPassword.text) & ")"

Function q(strng As String) As String
  '
  ' Данная функция возвращает передаваемую
  ' ей символьную переменную, заключенную
  ' в кавычки. Если символьная переменная
  ' содержит двойные кавычки, данная функция
  ' заключает эту переменную в одиночные
  ' кавычки.
  '
  If InSTR(1, strng, Chr(34)) Then
    q = Chr(39) & strng & Chr(39)
  Else
    q = Chr(34) & strng &Chr(34)
  End If
End Function
```

Эта функция экономит время и снижает возможность ошибок при написании длинных SQL-операторов. Единственный случай, когда она не работает, — наличие в одной и той же символьной переменной как одиночных, так и двойных кавычек. Однако это бывает крайне редко.

Андрей Колесов, Ольга Павлова

© 1997, Андрей Колесов, Ольга Павлова
Авторский вариант. Статья была опубликована с незначительной литературной правкой в журнале "КомпьютерПресс" N 5/97.

Совет 93. Используйте метод Refresh при работе с элементом управления SpinButton

(Совет прислал Вячеслав Найдич)

Элемент управления SpinButton ("кнопка-счетчик") представлен в виде двух стрелок (горизонтальных или вертикальных). Нажатие мышью одной из них генерирует событие SpinDown, другой — SpinUp. Обрабатывая эти события можно, например, дискретно изменять некоторое числовое значение в элементах TextBox, Label или положение текущего элемента в ListBox. В VB3 изменение значений в таких "присоединенных" объектах можно было увидеть сразу при нажатии соответствующей стрелки SpinButton, а в VB4 — только при отпускании кнопки.

Чтобы убрать этот эффект, в присоединенном объекте нужно использовать метод Refresh.

Пример:;

```
Private Sub SpinButton1_SpinDown()
  ' увеличение значения в TextBox
  Text2.Text = Str$(Val(Text2.Text) + 1)
  Label1.Caption = Str$(Val(Text2.Text) + 1)
  ' то же для Label Label1.Refresh
  ' для VB3 эта строка не нужна
End Sub

Private Sub Text2_Change()
  Text2.Refresh 'для VB3 эта строка не нужна
End Sub
```

TextBox имеет метод Change, поэтому вызов Refresh достаточно поставить в этом объекте один раз. Для Label его нужно ставить в самом объекте SpinButton в двух местах.

Совет 94. Обращение к процедуре формы

(Совет прислал Вячеслав Найдич)

Чтобы процедура Sub ProcName была доступна из любого места проекта, она должна быть объявлена как Public. Однако, если процедура реализована в модуле формы (например, Form2), правильное обращение к ней должно выполняться с указанием модуля, в котором она реализована. Например, если проект состоит из двух форм (без Module) и процедура, созданная в Form1, вызывается из Form2, то такой вызов выглядит так: Call Form2.ProcName (обращение как к объекту этой формы, а не просто Call ProcName).

Совет 95. Используйте свойство StartMode объекта App

В VB4 установка параметра StartMode во вкладке Project диалогового окна Options (меню Tools) определяет тип создаваемого модуля — Standalone или OLE Server. Фактически этот параметр определяет только, будет ли приложение, у которого нет формы запуска, выполняться после завершения процедуры Sub Main или нет. Этот параметр очень удобно применять для тестирования серверов OLE Automation.

Свойство StartMode объекта App позволяет задать на уровне кода режим показа пользователю какого-либо видимого интерфейса. С его помощью можно добиться, чтобы отдельный исполняемый VB-файл стал и невидимым сервером, и обычным приложением. Для этого в процедуре Sub Main следует проанализировать значение App.StartMode и решить, показывать форму или нет. Свойство StartMode равно vbSModeStandalone, если приложение запускается непосредственно пользователем, и vbSModeAutomation, если оно запускается клиентским приложением:

```
Sub Main
  If App.StartMode = vbSModeStandalone
    Then frmMain.Show
  Else
  ' запускается OLE сервер
  End If
End Sub
```

Совет 96. Используйте коллекцию Properties для объектов доступа к данным

В процессе отладки можно очень эффективно использовать коллекцию Properties, принадлежащую многим объектам доступа к данным. Для этого выполните следующий код из окна Debug:

```
For i = 0 to Recordset1.Properties.Count - 1
  Debug.Print Recordset1.Properties(i).Name & _
    Recordset1.Properties(i)
Next
```

Совет 97. Добавьте свойства к объектам доступа к данным

Вам необходимо, чтобы объекты типа Field имели свойство, определенное вами, например "RequiredIfCondition1"? Ваше желание легко осуществить с помощью такого кода:

```
Set NewProperty = Field1.CreateProperty("FieldNote")
NewProperty.Type = dbText
Field1.Properties.Append NewProperty
```

Совет 98. Нельзя создавать библиотеку OLE DLL, когда класс Public установлен равным Creatable — Single Use

В памяти в данный момент времени допускается наличие только одной копии DLL-библиотеки. Поэтому VB не разрешает создавать DLL-библиотеки, воздействующие на классы экземпляров единичного использования (single use instancing classes). Однако он позволяет создавать OLE-серверы, для которых можно многократно создавать экземпляры

для различных клиентов. Если вам нужен новый экземпляр сервера для каждого клиента, вы должны создать EXE- файл в форме сервера out-of-process вместо DLL-библиотеки.

Совет 99. Помните об учете регистра в названиях DLL функций

Как мы уже говорили ранее, в 32-разрядных DLL-библиотеках названия функций критичны к регистру написания букв: если вы вместо имени sndPlaySoundA напишите SndPlaySoundA, будет выдано сообщение об отсутствии такой функции в библиотеке WINMM.DLL нет. Для преобразования вызовов функций, используемых в VB 3.0 (там не было такого жесткого правила), в их 32-разрядные аналоги, которые зависят от регистра, можно применять ключевое слово Alias в операторе Declare Sub/Function. В файле Win32API.TXT записаны псевдонимы (Aliases) всех вызовов функций, которые помогут решить проблему учета регистра.

Совет 100. Используйте согласованное превращение букв в идентификаторах VB

Во избежание ошибок, связанных с неправильным написанием названий переменных (когда вместо NameFile вы случайно написали NaneFile и удивляетесь потом, что программа работает неправильно), мы настойчиво рекомендуем использовать режим Option Explicit (подробнее об этом см. КомпьютерПресс ь 5'96, Совет 9). Для тех, кто принципиально не желает применять эту опцию или не имеет такой возможности (например, работая в более ранних версиях MS Basic), можно посоветовать следующее.

Среда Basic поддерживает согласованность изменения написания имени любой переменной, приводя его в соответствие с последним вариантом написания этого имени. Другими словами, если вы используете переменную с именем myVariable, а затем напишите MyVariable, VB преобразует первое написание так, что оно будет совпадать со вторым.

Благодаря этому свойству можно убедиться в том, что имена переменных написаны правильно во всем коде программы. Напишите имя переменной целиком прописными буквами, например MYVARIABLE, и VB изменит все остальные его написания. Теперь если вы, просматривая свой код, вдруг обнаружите MyVarible, то будете знать, что это имя написано с ошибкой.

Совет 101. Создание двух версий одного проекта

Любой проект в VB 4.0 может иметь как MAK-, так и VBP-расширение. Данное свойство позволяет создавать две версии одного и того же проекта. Например, вы одновременно установили 16- и 32-разрядную версии VB 4.0. (Выбрав вариант "обе платформы", можно установить 16-разрядную версию в один каталог, а 32- разрядную — в другой.) Работая с обеими платформами, вы можете создать две версии одного проекта — MAK и VBP, которые будут совместно использовать формы и модули благодаря возможности условной компиляции. Для этого в диалоговом окне Make EXE File следует ввести разные имена исполняемых файлов и номера версий.

Совет 102. Опробуйте нижеследующие рекомендации по компиляции для защиты и оптимизации ваших программ

(Эти советы мы нашли в журнале VBPJ ь 14'96. Не все они кажутся нам очевидно правильными, но попробовать воспользоваться ими, наверное, имеет смысл.)

- Замените символьные константы на переменные и инициализируйте их в начале программы.
- Замените используемые больше одного раза символьные литералы на символьные константы.
- Не передавайте текстовые пароли в процедуры внутри или за пределами вашего кода. Вначале зашифруйте пароль.
- Если возможно, используйте массивы элементов управления, а не большое количество статических элементов управления одного типа.
- Сохраняйте свои формы и модули как ASCII-текст.
- Старайтесь, чтобы размер вашей формы не превышал 50 Кбайт в формате ASCII.
- Замените элементы управления, такие как командные кнопки, которые реагируют на события мыши, на элемент управления "изображение", содержащий внутри себя графическое изображение элемента управления.
- Объявите глобальные переменные, переменные на уровне модуля и переменные на уровне процедуры с помощью ключевого слова Status, чтобы уменьшить стековое пространство.
- После выгрузки формы всегда устанавливайте ее в значение равным Nothing, чтобы очистить данные в сегменте кода модуля этой формы.
- Выключите в явном виде элементы управления "таймер", когда ваше приложение уже готово. В противном случае родительская форма может не выгрузиться и таймер продолжит срабатывать.

Совет 103. Любителям программирования в кодах — вы можете читать P-код

Если вы принадлежите к людям, которые должны все увидеть своими собственными глазами, и хотите изучить Basic P-код, рекомендуем воспользоваться отладчиком низкого уровня, преобразующим исходный код в машинный. Попробуйте, например, VC++ CodeView для Windows или другую аналогичную утилиту. Все, что здесь необходимо, — один раз вызвать функцию API, DebugBreak.

Вот как это делается с помощью CodeView:

1. Загрузите CodeView для Windows при работе в среде VB.EXE. Для этого создайте значок для CodeView и в командной строке напишите следующее: cvw.exe vb.exe (укажите нужные пути файлов)
2. Поместите оператор отладчика в свой VB-код перед той строкой, которую вы хотите посмотреть:

```
3.   DebugBreak
4.   Text1.text = "Привет!"
```

5. Запустите вашу программу.

Когда выполнение кода дойдет до оператора DebugBreak, вы окажитесь в отладчике на команде INT 3, за которой будет следовать команда RET. Перейдете за RET — и вы окажетесь в интерпретаторе P-кода.

Теперь можно воочию убедиться, как много команд должно быть выполнено, прежде чем в окне редактирования появится слово "Привет!". Работа интерпретатора, как следует из его названия, состоит в том, чтобы просматривать информацию, находящуюся в EXE-файле, и интерпретировать ее в команды, понятные операционной системе.

Андрей Колесов, Ольга Павлова

© 199x, Андрей Колесов, Ольга Павлова
Авторский вариант. Статья была опубликована с незначительной литературной правкой в журнале "КомпьютерПресс" N 8/97, с. 141-147.

Совет 104. Проблемы с бета-версией VB 5.0

Судя по нашим контактам с читателями, многие из них уже познакомились с VB 5.0 на примере ее бета-версии. Соответственно, уже первого мая они обнаружили, что время, установленное для работы с ней, истекло (expired) и работа с программой заблокирована.

Однако главная неприятность заключается в следующем. Может возникнуть ситуация, когда перестанут работать и некоторые другие программы, установленные вместе с бета-версией на одном и том же компьютере. Дело в том, что блокировка запуска была направлена не только на саму программу VB, но и на ее компоненты, в том числе и на элементы управления OCX, общие для многих программ.

Самое неприятное, что при запуске такая программа (в нашем случае - программа оцифровки карт DIDGER) просто заявляла, что время ее работы истекло, не указывая конкретного модуля, в котором возникли проблемы.

В результате анализа ситуации выяснилось, что перестали работать модули COMDLG32.OCX и COMCTL32.OCX (о похожей проблеме с ними мы уже писали в Совете 88[5]). Причем самое интересное заключалось в том, что после удаления VB5 с компьютера эти файлы остались на диске, так как было определено, что они используются в других программах. В результате пришлось удалить данные файлы вручную, а потом записать их заново, переустановив программу DIDGER.

Следует иметь в виду, что, вполне возможно, заблокированными оказались и другие общие модули VB5.

Совет 105. При отладке не забывайте о сохранении проекта

Функции Windows API предоставляют программисту доступ к многочисленным расширенным функциям. Но опасность их использования заключается в том, что ответственность за корректность их применения полностью ложится на самого разработчика. Проблема же здесь заключается в том, что для изучения функций API зачастую требуется проведение серии экспериментов с ними. При этом весьма высока вероятность неверных действий со стороны программиста, в результате чего легко можно разрушить систему или тривиально "зависнуть".

[5] http://www.visual.2000.ru/develop/ms-vb/tips/9703.htm

Для обеспечения целостности своей работы при использовании функций API выберите команду Options из меню Tools, затем установите режим Save Before Run с подходящей для вас опцией Prompt либо Don't Prompt (запрашивать или не запрашивать подтверждение на сохранение проекта).

Совет 106. Контролируйте установленные на компьютере шрифты

Если вы указываете для метки или текстового окна какой-либо шрифт, не установленный на том компьютере, где выполняется данное приложение, Windows автоматически заменит его на другой шрифт, не сообщая об этом.

Однако если отсутствующий шрифт будет в явном виде назначен для объекта, то программа аварийно завершит свою работу, выдав сообщение об ошибке "Invalid property value".

Ниже предлагается простой способ, с помощью которого можно обнаружить, установлены ли используемые в вашей программе шрифты на данном компьютере. Этот прием будет полезен при написании программ, предназначенных для широкого распространения: вполне возможно, что могут встретиться случаи, когда на компьютере отсутствует даже шрифт Arial.

Вначале создайте массив невидимых меток на своем окне запуска, используя все шрифты, которые требуются для работы вашей программы. Для каждой метки установите два свойства FontName и Tag, например, следующим образом:

```
lblFont(0).FontName = "Century Gothic"
lblFont(0).Tag = "Century Gothic"

lblFont(1).FontName = "Symbol"
lblFont(1).Tag = "Symbol"
```

Если Windows подставит другой шрифт, то свойство FontName метки изменится на новый шрифт в процессе выполнения программы. Чтобы известить своих пользователей о том, что у них не хватает каких-либо шрифтов, вызовите следующую подпрограмму в начале своей программы:

```
Sub CheckFonts ()

  Dim crlf As String, i As Integer, temp As String
  ' Проверка, установлены ли необходимые шрифты
  For i = 0 To 1
    If lblFont(i).FontName <> lblFont(i).Tag Then
      temp = temp & ", " & lblFont(i).Tag
    End If
  Next

  If temp <> "" Then       ' Сообщение пользователю
    MsgBox "В Вашей системе " & _
      "должны быть установлены следующие шрифты:" & _
      Chr(13) & Chr(10) & temp, 48
  End If
End Sub
```

Замененный шрифт можно использовать с помощью свойства FontName той метки, которая содержит нужный шрифт (или сохранить FontName в глобальной переменной):

```
  ' Установка шрифта Century Gothic
  picText_display.FontName = lblFont(0).FontName
```

Совет 107. Новая утилита регистрации элементов управления

В состав Visual Basic 5.0 входит утилита, которая называется REGEDIT.EXE и находится на компакт-диске в каталоге VB50\Tools\RegUtils. Данное приложение позволяет регистрировать многочисленные OCX'ы на своем компьютере с помощью шаблонов файлов в качестве параметров командной строки. Например, чтобы зарегистрировать все OCX'ы в заданном каталоге, необходимо перейти в этот каталог и в командной строке написать REGEDIT *.OCX. Файл README, находящийся в каталоге \RegUtils, содержит инструкции для установки REGEDIT и двух других регистрирующих утилит.

Совет 108. Управление шириной заголовков столбцов

При динамическом изменении свойства ColumnHeaders элемента управления ListView в режиме выполнения программы обычно заранее ничего не известно о длине текста заголовка. Поэтому пользователь должен каждый раз заново настраивать ширину столбца при просмотре таблицы. Однако, создав метку, свойство Visible которой будет установлено равным "Ложь" (False), а свойство Autosize - "Истина" (True), вы можете поместить в эту метку текст, который будет затем использоваться в качестве заголовка. После этого используйте Label1.Width в аргументе Add для ColumnHeader:

```
Private Sub Command1_Click()
    Dim ColumnText as String, clmx as ColumnHeader
    ColumnText = "Очень длинный текст для" &_
         "элемента управления ListView"
    Label1.Caption = ColumnText
    Set clmx = ListView1.ColumnHeaders.Add_
         (, , ColumnText, Label1.Width)
    ListView1.View = lvwReport
End Sub
```

В других случаях может оказаться полезной автоматическая установка одинаковой ширины всех столбцов ListView. Это выполняется с помощью такого кода:

```
Sub ResizeListView(ListView As ComctlLib.ListView)
  Dim head As ColumnHeader
  With ListView
    For Each head In .ColumnHeaders
      head.Width = .Width / .ColumnHeaders.Count
    Next
  End With
End Sub
```

Андрей Колесов, Ольга Павлова

© 199x, Андрей Колесов, Ольга Павлова
Авторский вариант. Статья была опубликована с незначительной литературной правкой в журнале "КомпьютерПресс" N 12/97, с. 127-133.

Примечание. *В скобках указаны номера версий VB, для которых годятся данные советы.*

Совет 109. Используйте короткие имена файлов (VB4 32, VB5)

Если вы хотите обратиться из своей программы к какому-нибудь приложению, например редактору Paint, открыв его с указанным вами файлом, то будет лучше заменить его длинное имя на короткое. Такая операция имеет смысл в некоторых случаях: если, предположим, в имени файла есть пробелы, это может привести к некорректной работе редактора Paint. Для преобразования длинного имени в короткое можно использовать следующую процедуру, которая, в свою очередь, обращается к Win32 API:

```
Declare Function GetShortPathName Lib "kernel32" Alias _
     "GetShortPathNameA"(ByVal lpszLongPath As String, _
     ByVal lpszShortPath As String, ByVal cchBuffer As Long) As Long

Function ShortName(LongPath As String) As String
     Dim ShortPath As String, ret As String
     Const MAX_PATH = 260
     ShortPath = Space$(MAX_PATH)
     ret = GetShortPathName(LongPath, ShortPath, MAX_PATH)
     If ret Then ShortName = Left$(ShortPath, ret&)
End Function
```

Совет 110. Как обрезать полное имя файла до нужной длины (любые версии Basic)

В некоторых случаях бывает нужным вывести на экран полное имя файла, но при этом возникает проблема ограниченного пространства в заголовке формы, текстовом окне или метке. В этом случае часто применяется такой способ — если имя файла полностью выходит за некоторый предел, то выводится еще сокращенное представление, в котором внутренняя часть пути заменяется на многоточие. Например, вместо "C:\Programs\VisualB\Examples\Test.bas" будет выдаваться "C:\...\Examples\Test.bas".

Для выполнения такого преобразования имени файла можно использовать приведенную ниже процедуру. Предположим, что длина метки составляет 50 символов, тогда вы просто пишете:

```
Function LongDirFix$(Incoming%, Max%)
  Dim i%, LblLen%, StringLen%, TempString$
  TempString$ = Incoming$
  If Len(TempString$) <= Max% Then
    LongDirFix$ = TempString$: Exit Function
  End If
  LblLen% = Max% - 6
  For i% = Len(TempString$) - LblLen% To Len(TempString$)
     If Mid$(TempString$, i%, 1) = "\" Then Exit For
  Next
```

```
  LongDirFix$ = Left$(TempString$, 3) + "..." + _
     Right$(TempString$, Len(TempString$) - (i% - 1))
End Function
```

Совет 111. Используйте смысловые имена при работе с элементом управления DBGrid (VB4 16/32)

Столбцы в элементе управления DBGrid привязаны к полям базы данных, однако сам номер индекса столбца ничего не говорит о том, какое поле он представляет. Поэтому ссылка, например на DBGrid1.Columns(0).Value, сообщает не очень многое при работе с исходным текстом программы. Гораздо лучше, когда переменные имеют некоторые осмысленные имена, связанные с конкретными столбцами. Для этого в секции описаний для формы можно зарезервировать, например, такие переменные:

```
Dim ColOrder_ID As Column
Dim ColArticle_ID As Column
Dim ColAmount As Column
```

А в событийной процедуре Form_Load сделать следующие присвоения:

```
With DBGrid1
     Set ColOrder_ID = .Columns(0)
     Set ColArticle_ID = .Columns(1)
     Set ColAmount = .Columns(2)
End With
```

Теперь нет больше необходимости ссылаться на DBGrid1.Columns(0).Value — вместо этого вы можете использовать переменную ColOrder_ID.Value. Если меняется какое-либо значение свойства в DBGrid1.Columns(0), тогда аналогичные изменения произойдут в ColOrder_ID, и наоборот. Впоследствии, когда вы станете устанавливать связь между полями базы данных и сеткой, будет необходимо изменить только номера индексов столбцов в коде события Form_Load.

Совет 112. Структуры данных могут быть более сложными (VB4 16/32, VB5)

VB4 позволяет создавать более сложные структуры данных по сравнению с предыдущими версиями благодаря новой возможности, когда внутри типов могут содержаться динамические массивы. Например, следующая структура с тремя уровнями может быть создана с помощью заданных пользователем типов и динамических массивов:

```
Private Type LeafType
    ' пусть листья будут целыми переменными
    Leaf As Integer
End Type
Private Type BranchType
    ' ветвь представляет собой массив листьев
    Branch() As LeafType
End Type
' дерево - это массив ветвей
Private Tree() As BranchType
```

Следующий код задает дерево с тремя ветвями. На первой из них — три листа, на второй — два, а на третьей — пять:

```
ReDim Tree(1 To 3)                ' три ветви
ReDim Tree(1).Branch(1 To 3)      ' три листа на первой ветке
ReDim Tree(2).Branch(1 To 2)      ' два листа на второй ветке
ReDim Tree(3).Branch(1 To 5)      ' пять листьев на третьей ветке
```

Здесь можно добавить другие поля, чтобы другие части дерева, помимо листьев, хранили данные.

Совет 113. Быстрая проверка выходных дней (все версии Basic)

Прежде всего обратите внимание, что в Windows используется нумерация дней недели, отличающая от принятой в России: воскресенье — 1, понедельник — 2,.. Выполните следующее, чтобы проверить, не выпадает ли какая-либо дата на уик-энд (суббота, воскресенье):

```
nDay = weekday(sDate)  ' номер дня недели
If (nDay = 1) or (nDay = 7) Then
   ' это уик-энд
End If
```

Однако с помощью оператора MOD этот тест можно выполнить почти в два раза быстрее:

```
If (Weekday (sDate) MOD 6 = 1) Then
   ' это уик-энд
End If
```

Совет 114. Проверка високосного года (все версии Basic)

Вот еще один оригинальный способ проверки високосного года:

```
Function IsLeap (sYear As String) As Interger
  IsLeap = False
  If IsDate ("02/29/" & sYear) Then IsLeap = True
End Function
```

Совет 115. Динамическое управление положением элементов управления (VB4 16/32, VB 5)

Иногда возникает необходимость перемещения элементов управления на форме при изменении ее размеров. Для этого можно воспользоваться свойством Align-объекта. Например, попробуйте следующее:

```
Private Sub Form_Resize()
   Picture1.Align = vbAlignLeft
   Picture1.Align = vbAlignTop
End Sub
```

Каждое событие, которое изменяет размеры формы, автоматически перемещает рисунок в верхний левый угол. Перед выполнением процедуры убедитесь, что элемент управления, размеры которого вы хотите изменить, имеет свойство выравнивания (Align).

Совет 116. Используйте функции API для работы с файлами (все версии VB)

При работе в среде DOS мощным средством расширения встроенных языковых возможностей любой системы программирования, в том числе и Basic, является применение системных функций DOS/BIOS. С их помощью можно либо увеличить скорость выполнения операций по сравнению с использованием операторов языка, либо расширить функции операторов, либо получить в свое распоряжение совершенно новые функции.

С точки зрения программиста принципиальным отличием функций DOS/BIOS от других библиотек процедур было то, что этот набор был всегда под рукой (в составе DOS) и не требовал специальных операций по его подключению к программе. Кроме того, для обращения к нему использовалась одна входная точка CALL INTERRUPT, а дальнейшее определение операции и передача параметров выполнялись довольно специфичным заполнением целочисленной структуры данных. (На эту тему у авторов было довольно много публикаций, электронный вариант которых хранится у них.)

При работе в среде Windows таким средством расширения языка является набор WinAPI, многие процедуры которого являются непосредственными аналогами функций DOS/BIOS. По сути дела, последние так и остались в операционной системе, но обращение к ним производится не напрямую через системное прерывание, а через WinAPI.

Одна из групп системных функций DOS под названием "Handle-ориентированный ввод-вывод" была связана с работой с файловой системой. Фактически эти функции соответствовали операторам Basic при работе с двоичными файлами Binary, но имели целый ряд важных преимуществ. Например, операторы Get/Put могут за одну операцию передавать только одну простую переменную. Соответственно, для записи массива на диск нужно создавать некоторый цикл обращений к файлу:

```
Dim Array (1 To N%) As Single
...
For i% = 1 To N: Put #FileNumber,,Array(i): Next
```

При работе с функциями DOS на диск записывалась некоторая непрерывная последовательность байтов, для этого указывался начальный адрес этой области (сегмент и смещение) и ее длина. Таким образом, можно было, например, записать в файл за одно обращение к диску весь массив целиком (при работе с фрагментами многомерных массивов следует учитывать физическое хранение его элементов). Приведенный выше пример можно выполнить следующим образом с использованием функций DOS :

```
regsx.ax = &H4000            'операция записи
regsx.bx = FileHandle%       'описатель файла
regsx.ds = VARSEG(Array(1))  'адрес сегмента
regsx.dx = VARPTR(Array(1))  'смещение
regsx.cx = N% * 4            'число байтов
CALL interruptx(&H21, regsx, regsx)
```

Примечание. *На самом деле определение числа байтов для переменной regsx.cx нужно выполнять по-другому. Об этом — см.* [*совет 117*](#)[6].

[6] http://www.visual.2000.ru/develop/ms-vb/tips/9712.htm

В зависимости от размера массива скорость обмена данными могла возрастать в сотни раз. Следует также подчеркнуть, что с помощью функций DOS группы Handle можно было выполнять ряд операций, в принципе не реализуемых операторами Basic, например, переопределение стандартных устройств ввода-вывода. Поэтому в своих более ранних разработках на Basic/DOS мы часто использовали библиотеки подпрограмм, основанные на использовании этих функций DOS/Handle.

При работе с VB реализация таких расширенных Handle-функций возможна с помощью функций WinAPI: lclose, lcreate, llseek, lopen, lread, lwrite. Кроме того, еще в наборе API для Windows 3.1 появились две дополнительные функции — hwrite и hread, которые в отличие от процедур lread и lwrite (и соответствующих функций DOS) могут выполнять обмен массивами данных размером более 64 Кбайт. Все эти функции имеются как для WinAPI-16 (файл "Kernel"), так и для WinAPI-32 ("Kernel32"). Отличие этих двух наборов функций заключается в том, что в первом случае числовые параметры передаются в виде переменных Integer, а во втором — Long.

Приведенная ниже процедура WriteIntegerArrayToFile является примером записи двумерного целочисленного массива i_array в двоичный файл binary_file с помощью API-функции hwrite (вариант WinAPI-16). Данная процедура легко модифицируется для других размеров и типов массивов (за исключением массивов, определенных пользователем, и строк переменной длины). Тем не менее ее невозможно преобразовать в общую процедуру для любого числа индексов массива из-за способа вызова API-функций. Следует также обратить внимание на следующие моменты: 1. В функциях DOS в качестве параметра передавался адрес области памяти в виде двух целочисленных переменных. В функциях API передается некая переменная, которая на самом деле представляет именно ссылку на ее физический адрес. 2. Вывод данных выполняется в текущую позицию файла. Управление текущей позицией файла производится функцией llseek. 3. При работе с файлами на Basic используется понятие "логический номер файла", а при использовании функций DOS/API — "номер описателя файла" (handle). Следует обратить внимание, что это — разные величины. Для определения номера описателя файла в DOS по его логическому номеру используется оператор FileAttr. Таким образом, можно совмещать одновременную работу с открытым файлом средствами как Basic, так и DOS/API.

```
' Запись массивов в двоичный файл:
Declare Function hwrite Lib "Kernel" Alias "_hwrite" _
     (ByVal hFile As Integer, hpbBuffer As Any, _
     ByVal cbBuffer As Long) As Long
' В качестве значения функции возвращается
' число переданных байтов
'
Sub WriteIntegerArrayToFile (ByVal binary_file As String, _
     integer_array() As Integer)
Const INTEGER_BYTE_SIZE& = 2  ' число байтов в переменной
Dim binary_file_number As Integer, dos_file_handle As Integer
Dim bytes_written As Long, bytes_to_write As Long
Dim start1 As Integer, start2 As Integer
     ' индексы первого (физически) элемента массива
     start1 = (LBound(i_array, 1)
     start2 = (LBound(i_array, 2)
     ' Получение размера массива в байтах (произведение числа строк на
     ' число столбцов и на число байтов в одном элементе массива)
     bytes_to_write = INTEGER_BYTE_SIZE& * _
     (UBound(i_array, 1)
- start1 + 1) * _
     (UBound(i_array, 2) - start2 + 1)
     ' открытие двоичного файла
```

```
     binary_file_number = FreeFile   ' логический номер
     Open binary_file For Output As binary_file_number
     ' получаем номер описателя файла в DOS
     dos_file_handle = FileAttr(binary_file_number, 2)
     ' вызов API-функции
     If dos_file_handle <> 0 Then
       bytes_written = hwrite(dos_file_handle, _
         integer_array (start1, start2), bytes_to_write)
     End If
     ' закрыть двоичный файл
     Close binary_file_number
End Sub
```

На самом деле кроме упомянутых выше процедур в составе WinAPI имеется много других функций, которые не имеют аналогов среди функций DOS. Например, это целый набор операций с INI- файлами, управление версиями файлов (Version Stamping), сжатыми файлами и пр. Но есть и старые функции DOS, которые не реализованы в API.

Совет 117. Работа с беззнаковыми целыми числами (все версии Basic)

Одним из явных недостатков всех версий Microsoft Basic является отсутствие в них беззнаковых целочисленных переменных с диапазоном изменения 0-65 535/&hFFFF (такой тип переменных был во всех версиях Turbo-BASIC, PowerBASIC еще десятилетней давности). Проблема заключается в том, что именно такой тип данных очень широко применяется на системном уровне. При работе в среде DOS это четко проявлялось при использовании системных функций DOS/BIOS, в частности, при представлении адреса в виде двух целочисленных регистров, и наоборот. Аналогичная проблема имеется и при работе в Windows. Например, при использовании функций lwrite/lread в WinAPI-16, как и при работе с аналогичной функцией DOS (см. предыдущий совет), число записываемых данных передается в виде целочисленной переменной, которая на самом деле может изменяться от 0 до 65 535.

Для подобных преобразований могут пригодиться следующие процедуры:

```
SUB ConvertIntegersToLong (LongValue&, IntegerHigh%, IntegerLow%)
'   Представление двух беззнаковых переменных типа INTEGER
'   в виде переменной типа LONG
    LongValue& = IntegerHigh% * &H10000 + (IntegerLow% AND &H7FFF)
    IF IntegerLow% < 0 THEN LongValue& = LongValue& OR 32769 ' &h8000
END SUB

SUB ConvertLongToIntegers (LongValue&, IntegerHigh%, IntegerLow%)
'   Представление переменной типа LONG в виде двух
'   беззнаковых переменных типа INTEGER
'
    IntegetHigh% = LongValue& \ &H10000   'старшие 16 разрядов
    IntegerLow% = LongValue& AND &H7FFF   'младшие 16 разрядов
    IF (LongValue& AND &H8000) <> 0 THEN
      IntegerLow% = IntegerLow% OR &H8000
    END IF
END SUB
```

В этом случае приведенное ранее вычисление длины передаваемого массива должно выглядеть следующим образом:

```
' ВМЕСТО: regsx.cx = N% * 4            'число байтов
Call  ConvertLongToIntegers (N% * 4&, IntegerHigh%, regsx.cx)
```

```
If IntegerHigh% <> 0 Then
  ' превышение допустимой длины
End If
```

При работе в Win32 такие проблемы возникают гораздо реже, так как для передачи целочисленных данных там изначально используются переменные типа Long.

Совет 118. Как определить длину файла (все версии VB)

Как было сказано в совете 116, управление указателем текущей позиции файла производится с помощью функции API llseek, которая для 16-разрядной версии описывается следующим образом:

```
Declare Function llseek Lib "Kernel" Alias "_llseek" _
               (ByVal hFile%, ByVal lOffset&, ByVal iOrigin%) As Long
```

где Offset& — на сколько передвинуть указатель (Long), а iOrigin% — точка отсчета (Integer): 0 — начало файла, 1 — текущая позиция, 2 — конец файла. Результатом функции является новое значение текущей позиции, например:

```
NewSeek& = llseek (hFile%, 2, 0) 'определение длины файла
NewSeek& = llseek (hFile%, 1, 0) 'определение текущей позиции
```

Андрей Колесов, Ольга Павлова

© 199x, Андрей Колесов, Ольга Павлова
Авторский вариант. Статья была опубликована с незначительной литературной правкой в ....

В этот выпуск мы включили несколько советов, связанных с перекодировкой строковых переменных.

Совет 119. Используйте функции API для перекодировки текста.

При решении многих задач требуется выполнить перекодировку текста из одной кодовой таблицы в другую, например из 866 в 1251, или наоборот. Чтобы справиться с этой проблемой, можно обратиться к соответствующим функциям API.

Напомним, что Windows 3.x/95 использует в своей работе две кодовые таблицы, которые мы обычно называем DOS и Windows, но формально они именуются OEM и ANSI. Именно последние названия используются в соответствующих именах параметров управляющих файлов Windows. Для отечественных пользователей эти таблицы имеют номера 866 и 1251 соответственно. (Чтобы не вводить своих пользователей в искушение, Microsoft не позволяет производить динамическое переопределение номеров кодовых таблиц — они устанавливаются только при инсталляции ОС. Впрочем, смельчаки могут отважиться и на это, непосредственно изменив параметры управляющих файлов, что вполне возможно, но... лучше не играть с огнем.)

В составе API есть процедуры преобразования кодировки из OEM-таблицы в ANSI, и наоборот. В 32-разрядном наборе интерес представляют прежде всего следующие функции:

CharToOem — преобразование строковой переменной из ANSI в OEM (в API/16 аналогичная функция называется AnsiToOem);

OemToChar — преобразование строковой переменной из OEM в ANSI (API/16 — OemToAnsi);

В API 16/32 есть и другие функции, связанные с преобразованием символьных данных. Их применение можно проиллюстрировать таким примером (API/32):

```
Private Declare Function GetOEMCP Lib "kernel32" () As Long
Private Declare Function GetACP Lib "kernel32" () As Long
Private Declare Function OemToChar Lib "user32" Alias "OemToCharA"
(ByVal lpszSrc As String, ByVal lpszDst As String) As Long

Private Sub Form_Load()
   ' сначала можно проверить номера кодовых таблиц
   OemCP& = GetOEMCP    ' номер OEM (DOS) таблицы
   AnsiCP& = GetACP     ' номер ANSI (Windows) таблицы
   Debug.Print OemCP&; AnsiCP&   ' = 866 2151
   '
   ' открывает файл с русским текстом
   FilePath$ = "russian.txt"
   Open FilePath$ For Input As #1
   Line Input #1, InputStr$
   Print InputStr$
   OutputStr$ = Space$(Len(InputStr$)) ' ВНИМАНИЕ НА ЭТУ СТРОЧКУ
   Code& = OemToChar(InputStr$, OutputStr$)  '
   '
   Print Code&: Print InputStr$: Print OutputStr$
   Close #1
End Sub
```

Здесь нужно обратить внимание на следующие моменты:

1. Данная функция производит перекодировку из установленной кодовой таблицы DOS в таблицу Windows. Если вы пользуетесь американской Windows OSR2, то в ней, скорее всего, не установлены страницы 866 и 1251.
2. Переменная OutputStr$, в которую записывается результат операции, должна быть сформирована до обращения к функции OemToChar. Если она вообще не была создана (в случае, например, комментария в строке OutputStr$=...), то выдается код Code& = 0. Если данная переменная существует, то получаем Code& = 1. Но при этом формируется только то число символов, которое было изначально определено. Для эксперимента установите: OutputStr$ = Space$(0) или OutputStr$ =Space$(Len(InputStr$)) + "ffffffff".
3. Если вы не боитесь потерять исходное значение переменной, то проще использовать одну и ту же переменную: Code& = OemToChar(InputStr$, InputStr$).

Совет 120. Простые процедуры преобразования символьных данных.

Проблемы перекодировки не ограничиваются только преобразованием из таблиц DOS в Windows. Для их решения хотелось бы предложить некоторые простые процедуры, которые сохранились у нас еще со времен QuickBasic 4.x:

```
FUNCTION SymChange$ (Text$, NewCode$, OldCode$)
'
'  Перекодировка символов  OldCode$ -> NewCode$:
'  в переменной Text$ все символы набора OldCode$
'  заменяются на соответствующие символы набора NewCode$
```

```
'
'  ВНИМАНИЕ!
'  Должно выполняться условие:  LEN(NewCode$)=LEN(OldCode$)
'_________________-
     Sym$ = Text$: Ltext% = LEN(Text$)
     IF Ltext% > 0 THEN
       FOR i% = 1 TO Ltext%
         k% = INSTR(OldCode$, MID$(Text$, i%, 1))
         IF k% > 0 THEN
           MID$(Sym$, i%, 1) = MID$(NewCode$, k%, 1)
         END IF
       NEXT i%
     END IF
     SymChange$ = Sym$
END FUNCTION

FUNCTION RusDosWin$ (Text$, Code%)
'
'  Преобразование КИРИЛЛИЦЫ:
'  Code% = 0 - из DOS в Windows
'        = 1 - обратно
'_____________
'  D$ - константа кодов русских символов для DOS (cp866)
'  W$ - константа кодов русских символов для Windows (cp1251)
'  обе константы содержат такую последовательность символов:
'  АБВ...Яабв...я
'
'=============================================
CONST D$ = "ЂЃ‚ѓ„…†‡€‰Љ‹ЊЌЋЏ‘’“”•–—™љ›њќћџ_
 ЎўJ¤Ґ¦§Ё©Є«¬®Їабвгдежзийклмноп"
CONST W$ = "АБВГДЕЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯ_
абвгдежзийклмнопрстуфхцчшщъыьэюя"
' ============================================
IF Code% = 0 THEN
      RusDosWin$ = SymChange(Text$, W$, D$)
    ELSE
      RusDosWin$ = SymChange(Text$, D$, W$)
    END IF
END FUNCTION
FUNCTION ULrus$ (Text$, Code%)
'
'   Преобразование русских символов:
'   Code% = 0  - строчные в прописные
'         = 1  - наоборот
'         ELSE - ничего не делаем
'   (Аналог операторов UCASE$/LCASE$, но для русских символов)
'_______________
     CONST Urus$ = "ЙЦУКЕНГШЩЗХФЫВАПРОЛДЖЭЯЧСМИТЬБЮЪ"
     CONST Lrus$ = "йцукенгшщзхфывапролджэячсмитьбюъ"
'
     SELECT CASE Code%
      CASE 0: ULrus$ = SymChange$(Text$, Urus$, Lrus$)
      CASE 1: ULrus$ = SymChange$(Text$, Lrus$, Urus$)
     END SELECT
END FUNCTION

FUNCTION DoubleSpaceDelete$ (Text$)
'
'   Убирает двойные пробелы в переменной Text$
'___________
  Temp$ = RTRIM$(Text$)
  DO
    Num% = INSTR(Temp$, "  ")  ' поиск двойного пробела
```

```
    IF Num% <= 0 THEN EXIT DO
    Temp$ = LEFT$(Temp$, Num%) + MID$(Temp$, Num% + 2)
  LOOP
  DoubleSpaceDelete$ = Temp$
END FUNCTION

FUNCTION StrFilter$ (Text$, Filter$)
'=============================================================
'   Фильтрация содержимого Text$
'
'   Filter$=CHR$(CodeFilter%)+FilterString$
'   Первый символ переменной Filter$ - код операции,
'   остальные символы - перечень допустимых/НЕдопустимых символов
'   CodeFilter%=ASC(Filter$)
'      =0 - удаление всех символов из Text$, кроме указанных в
'               списке допустимых - FilterString$
'      =1 - удаление всех символов из Text$, указанных в
'               списке НЕдопустимых - FilterString$
'=============================================================
      Temp$ = "": CodeFilter% = ASC(Filter$)
      FOR i% = 1 TO LEN(Text$)
        Sym$ = MID$(Text$, i%, 1)
        IF (SGN(INSTR(2, Filter$, Sym$)) XOR CodeFilter%) <> 0 THEN
          Temp$ = Temp$ + Sym$
        END IF
      NEXT i%
      StrFilter$ = Temp$
END FUNCTION
```

Эти процедуры представлены в формате Basic/DOS (операторы языка написаны прописными буквами) и наглядно демонстрируют преемственность исходных текстов версий DOS/Windows — для VB/Win нужно только удалить операторы DECLARE.

1. Функция StrChange$. С ее помощью можно производить замену того или иного набора символов на любой другой. Например, в строке можно поменять местами буквы "т" и "м":

```
a$ = "там"
b$ = SymChange$(a$, "тм", "мт")
PRINT b$  ' напечатано - "мат"
```

5. Функции RusDosWin$ и ULrus$ представляют собой примеры практического применения функции StrChange$. Первая производит преобразование русских символов из кодировки DOS в Windows и наоборот, а вторая — строчных русских символов в прописные, и наоборот. Набор русских символов в кодировке DOS и Windows (константа W$) очень просто получить программным образом:

```
W$ = "": D$ = ""
For i = 192 To 255: W$ = W$ + Chr$(i): Next
For i = 128 To 175: D$ = D$ + Chr$(i): Next
For i = 224 To 236: D$ = D$ + Chr$(i): Next
Print W$: Print D$
```

11. Функция DoubleSpaceDelete$ убирает лишние пробелы в строке (в качестве разделителя слов остается один пробел).
12. Функция StrFilter$ удаляет допустимые или недопустимые символы из строковой переменной в соответствии с заданным пользователем набором символов. Например, с ее помощью можно убрать все разделители из номера телефона, чтобы представить его в чисто цифровом виде:

```
PhoneNumber$ = "369-76-97"
CodeFilter% = 0       ' код операции
FilterString$ = "1234567890"
Filter$ = CHR$(CodeFilter%) + FilterString$
NewNumber$ = StrFilter$(PhoneNumber$, Filter$)
PRINT NewNumber$
```

```
19.  ' напечатано: '3697697'
```

Совет 121. Будьте внимательны при преобразовании исходных текстов программ из DOS в Windows, и наоборот.

Многие процедуры, написанные в версиях Basic для DOS, могут применяться и в VB/Win. Но при переносе исходных программных модулей из DOS в Windows (и наоборот) нужно быть очень внимательным. И особенно в том случае, если вы хотите, чтобы одна и та же копия модуля могла использоваться в обеих версиях. Проблема здесь заключается в следующем: DOS и Windows используют разные кодировки русских букв — таблицы с номерами 866 и 1251 соответственно.

Если напрямую загрузить написанный в DOS модуль в VB, то очевидно, что русский текст, например комментарии, превратится в нечитаемый набор символов. Если вы работаете только в VB/Win, то такой исходный файл, конечно же, лучше сначала преобразовать из кодировки 866 в 1251. Для этого можно, к примеру, использовать WinWord: прочитать исходный файл как "Текст MS-DOS", а запомнить как "Только текст". Впрочем, если вопрос касается только тех комментариев, которые вам не очень-то и нужны, то такое преобразование можно не делать, а работать с одним исходным файлом как в DOS, так и в Windwos.

Другое дело, если русские символы используются непосредственно в коде программы. Самый простой пример: Print "Привет, Коля!" В этом случае перекодировка исходного текста процедуры является просто необходимой. Как следствие, одну и ту же копию программного модуля, использующего подобные символьные константы, нельзя применять при работе и в DOS и в Windows.

В то же время может возникнуть и другая ситуация, примером которой является текст приведенной выше функции RusDosWin$. Дело в том, что используемые в ней константы D$ и W$ являются не осмысленным текстом, а кодовыми таблицами, представленными для удобства в символьном виде. Каждая константа содержит коды символов последовательности "АБ...Яаб...я" для кодировки DOS и Windows соответственно. Поэтому, если смотреть текст процедуры RusDosWin$ в среде DOS, вы должны увидеть:

```
Const D$ = "АБ...Яаб...я"
Const W$ = хаотичный набор символов
```

а если в Windows, то наоборот:

```
Const D$ = хаотичный набор символов (но другой!)
Const W$ = "АБ...Яаб...я"
```

При этом вас не должен смущать появляющийся в изображении хаотичный набор символов повторяющихся знаков — просто для изображения разных кодов, не используемых реально для вывода, применяются одни и те же знаки.

Хотелось бы обратить внимание, что при написании процедуры RusDosWin$ в среде DOS в редакторе QB была введена с клавиатуры только константа D$, а константа W$ появляется в нем совершенно иным способом. Для этого в "Лексиконе>"для DOS был сделан отдельный текстовый файл, содержащий D$. Затем этот файл прочитали в WinWord как "Текст MS-DOS", сделали единственное изменение — D$ на W$ — и сохранили как "Только текст". Прочитав новый файл в DOS, можно увидеть, что он

содержит константу W$. Если же набирать текст в Windows, то делается аналогичная операция преобразования, но при этом с клавиатуры вводится уже содержимое W$.

Из вышесказанного можно сделать два вывода:

1. Нельзя преобразовывать исходный текст процедур типа RusDosWin$ из одной кодировки в другую. Вернее так: можно преобразовывать комментарии, нужно преобразовывать константы типа "Привет, Коля!" (это касается, например, констант Urus$ и Lrus$ в процедуре ULrus$), но нельзя преобразовывать специальные кодовые последовательности, представленные в виде символьных констант. Соответственно, перевод файла с процедурой RusDosWin$ из DOS в Windows необходимо сделать в два этапа: весь текст перекодировать обычным образом, а потом перенести в него строки с D$ и W$ без перекодировки.
2. Если вы захотите создать файл с процедурой RusDosWin$ из приведенного выше листинга, то можно ввести с клавиатуры только константу D$ (при работе в DOS) или W$ (в Windows), а вторую константу создать самостоятельно, выполнив, например, приведенную выше операцию.

Примечание. *Чтобы разобраться, почему при распечатке листинга разным символам в кодировке DOS соответствуют одни и те же символы в кодировке Windows, наберите в редакторе для DOS файл с тестом:*

```
АБВГДЕЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯ
```

а затем введите его в WinWord в режиме "текст" (без преобразования кодов) и сохраните его в том же режиме "текст". Просмотрев его в DOS, вы увидите:

```
АБВГДЕЖЗИЙКЛМНОПР''""ХЦЧШЩЪЫЬЭЮЯ
```

Такое несоответствие объясняется очень просто: символы с кодом 145 и 146 (в DOS — C и T) означают левую и правую одиночную кавычку, которые при записи в режиме "текст" заменяются на стандартную одиночную кавычку с кодом 39. А все потому, что в DOS был один код кавычки, а в Windows — три (старая оставлена для совместимости)! То же самое происходит и с двойной кавычкой (34, 147, 148).

Андрей Колесов, Ольга Павлова

© 1998, Андрей Колесов, Ольга Павлова
Авторский вариант. Статья была опубликована с незначительной литературной правкой в журнале "КомпьютерПресс" N 3/98, с.194-195.

Совет 122. Используйте метод GetRows для сохранения полей базы данных в переменных

Иногда возникает необходимость переписать в переменные информацию, хранящуюся в полях базы данных. Это можно сделать с помощью метода GetRows объекта Recordset, который копирует одну или несколько строк данных непосредственно в переменные типа Variant и хранит эту информацию в виде двухмерного массива в formvarData(Field,Column).

Протестируем метод GetRow. Для этого поместим командную кнопку на VB-форму и введем следующий код в событие Click. Но прежде необходимо изменить путь к базе данных BIBLIO.MDB в методе OpenDatabase в соответствии с вашими установками, а также задать ссылку на Microsoft DAO 3.5 Object Library.

```
Private Sub cmdGetDataRow_Click()
    '
   ' Пример использования метода GetRow
    '
    Dim ws As Workspace, db As Database, rs As Recordset
    '
    Dim varDataRows As Variant, strMsg As String
    Dim intRows As Integer, intColumns As Integer
    '
    Dim intLoopRow As Integer, intLoopCol As Integer,
    '
    Set ws = DBEngine.CreateWorkspace(App.EXEName, "admin", "")
    Set db = ws.OpenDatabase("e:\devstudio\vb\biblio.mdb")
    Set rs = db.OpenRecordset("SELECT * FROM Authors")
    '
    intRows = InputBox("Введите количество строк", _
        "Пример использования метода GetRows", 0)
    intColumns = rs.Fields.Count
    varDataRows = rs.GetRows(intRows)
    '
    For intLoopRow = 0 To intRows - 1
        strMsg = ""
        For intLoopCol = 0 To intColumns - 1
            strMsg = strMsg & varDataRows(intLoopCol, intLoopRow) & vbCrLf
        Next
        MsgBox strMsg
    Next
    '
    rs.Close
    db.Close
    ws.Close
    '
End Sub
```

Совет 123. Вывод длинных элементов списка в виде подсказки ToolTip

Довольно часто случается, что на форме слишком мало места для размещения окна списка, которое бы полностью выводило свои элементы. В этом случае можно воспользоваться следующим простым кодом, с помощью которого элементы списка выводятся в виде подсказок ToolTip, если вы помещаете курсор мыши на окно списка.

Вначале создайте новый VB-проект и поместите на форму элемент управления ListBox. Затем объявите вызов функции SendMessage API и константу LB_ITEMFROMPOINT, необходимые для выполнения нашей задачи.

```
Option Explicit

' Объявляет вызов функции API
Private Declare Function SendMessage Lib "user32" Alias "SendMessageA" _
  (ByVal hwnd As Long, ByVal wMsg As Long, ByVal wParam As Long, _
  lParam As Any) As Long
' Добавляет константу API
Private Const LB_ITEMFROMPOINT = &H1A9
```

После этого введите следующий код в событие Load для формы, чтобы заполнить окно списка:

```
Private Sub Form_Load()
    '
    ' Заполнение окна списка
    With List1
        .AddItem "Программирование баз данных в Visual Basic 5"
        .AddItem "Excel Visual Basic для разработки приложений"
        .AddItem "Руководство по VBA для MS Office 97"
    End With
End Sub
```

И наконец, в событии MouseMove элемента управления ListBox введите такой код:

```
Private Sub List1_MouseMove(Button As Integer, _
    Shift As Integer, X As Single, Y As Single)
    '
    ' Выводит соответствующую подсказку TipTool
    '
    Dim lXPoint As Long
    Dim lYPoint As Long
    Dim lIndex As Long
    '
    If Button = 0 Then ' если ни одна кнопка не была нажата
        lXPoint = CLng(X / Screen.TwipsPerPixelX)
        lYPoint = CLng(Y / Screen.TwipsPerPixelY)
        '
        With List1
         ' Выбирает элемент списка
         lIndex = SendMessage(.hwnd, LB_ITEMFROMPOINT, 0, _
              ByVal ((lYPoint * 65536) + lXPoint))
            ' Выводит новую подсказку или стирает старую
            If (lIndex >= 0) And (lIndex <= .ListCount) Then
                .ToolTipText = .List(lIndex)
            Else
                .ToolTipText = ""
            End If
        End With '(List1)
    End If '(button=0)
    '
End Sub
```

Совет 125. Простая проверка ввода данных

Здесь мы покажем, как организовать проверку в текстовых окнах или других элементах управления, которые поддерживают событие KeyPress. Такая проверка очень проста, но во многих случаях она бывает крайне необходима.

Вначале добавьте следующую функцию к своему проекту:

```
Function ValiText(KeyIn As Integer, ValidateString As String, _
   Editable As Boolean) As Integer
    '
    Dim ValidateList As String
    Dim KeyOut As Integer
    '
    If Editable = True Then
         ValidateList = UCase(ValidateString) & Chr(8)
    Else
```

```
         ValidateList = UCase(ValidateString)
    End If
    '
    If InStr(1, ValidateList, UCase(Chr(KeyIn)), 1) > 0 Then
        KeyOut = KeyIn
    Else
        KeyOut = 0: Beep
    End If
    '
    ValiText = KeyOut
    '
End Function
```

Затем для каждого элемента управления, где вы хотите организовать проверку, введите в событие KeyPress код наподобие следующего:

```
KeyAscii = ValiText(Keyascii, "0123456789/-",True)
```

Он отфильтрует любые нежелательные нажатия клавиш, оставив только те, которые задаются с помощью второго параметра. В нашем примере параметр "0123456789/-" определяет символы, которые могут использоваться для ввода даты.

При помощи третьего параметра функции вы можете управлять клавишей Backspace, то есть разрешать или запрещать ее использование.

Обратите внимание, что в данном примере регистр клавиатуры не влияет на проверку ввода данных. Например, если второй параметр равен "абвгдежз", то вы также сможете вводить и "АБВГДЕЖЗ".

Совет 126. Управление длиной элемента списка ComboBox

Элемент управления ComboBox, в отличие от текстового окна, не имеет свойства MaxLength. Однако нет ничего проще, чем добавить недостающее свойство. Для этого достаточно ввести в событие KeyPress элемента управления ComboBox следующий код:

```
Private Sub Combo1_KeyPress(KeyAscii As Integer)
    '
    ' Если пользователь попытается нажать одиннадцатую клавишу и
    ' если эта клавиша не Backspace, то отменить данное событие
    '
    Const MAXLENGTH = 10
    If Len(Combo1.Text) >= MAXLENGTH And KeyAscii <> vbKeyBack Then
      KeyAscii = 0
    End If
End Sub
```

Константа MaxLength может иметь любое значение. Кроме того, вместо Backspace вы можете использовать любые другие клавиши. Для этого просто введите их значения KeyAscii, как показано в примере с клавишей Backspace.

Андрей Колесов, Ольга Павлова

© 1998, Андрей Колесов, Ольга Павлова
Авторский вариант. Статья была опубликована с незначительной литературной правкой в журнале "КомпьютерПресс" N 7/98, с.194-196.

Совет 127. Выводите осмысленные сообщения об ошибках при вызовах функций Win32 API

Большинство функций Win32 API выводит на экран сообщение об ошибках, возникающих при их некорректном использовании. Чтобы сделать эту информацию более содержательной, следует воспользоваться функциями GetLastError и FormatMessage.

Для этого введите в BAS-модуль VB-проекта следующие описания и функцию:

```
Option Explicit

Public Declare Function GetLastError Lib "kernel32" () As Long
Public Declare Function FormatMessage Lib "kernel32" Alias _
  "FormatMessageA" (ByVal dwFlags As Long, lpSource As Any, _
  ByVal dwMessageId As Long, ByVal dwLanguageId As Long, _
  ByVal lpBuffer As String, ByVal nSize As Long, _
  Arguments As Long) As Long

Public Const FORMAT_MESSAGE_FROM_SYSTEM = &H1000

Public Function LastSystemError() As String
    '
    ' системная ошибка
    Dim sError As String * 500
    Dim lErrNum As Long
    Dim lErrMsg As Long
    '
    lErrNum = GetLastError
    lErrMsg = FormatMessage(FORMAT_MESSAGE_FROM_SYSTEM, _
      ByVal 0&, lErrNum, 0, sError, Len(sError), 0)
    LastSystemError = Trim(sError)
    '
End Function
```

Затем поместите командную кнопку на стандартную VB-форму и вызовите функцию LastSystemError:

```
Private Sub Command1_Click()
    '
    MsgBox LastSystemError
    '
End Sub
```

Если не было зарегистрировано никакой ошибки, вы увидите сообщение со словами "The operation completed successfully".

При использовании данной функции следует помнить следующее:

1. Большинство вызовов функций API при успешном выполнении операции сбрасывает значение GetLastError в первоначальное состояние, поэтому функцию LastSystemError следует вызывать сразу же после того вызова функции API, который создает ошибочную ситуацию.
2. Значение последней ошибки хранится отдельно для каждого процесса, поэтому функцию LastSystemError необходимо вызывать из того же процесса, откуда осуществлялся вызов функции API, приводящий к ошибке.

Совет 128. Увеличение и уменьшение даты с помощью клавиш [+] и [-]

В некоторых программах работа с датой реализована довольно интересным образом. Нажатие клавиши [+] увеличивает дату на один день, клавиши [-] — уменьшает на один день, клавиша [PgDn] прибавляет один месяц, а клавиша [PgDn] убавляет на один месяц. Попробуем реализовать это с помощью VB.

Вначале поместите на форму элемент управления TextBox (txtDate). Установите его свойство Text равным "", а свойство Locked — True. После этого введите следующий код в событие KeyDown:

```
Private Sub txtDate_KeyDown(KeyCode As Integer, Shift As Integer)
    '
    ' KeyCode — специальный код клавиши (а не ASCII-код!)
    ' 107 = "+" KeyPad (цифровая клавиатура)
    ' 109 = "-" KeyPad
    ' 187 = "+" (в действительности это клавиша "=")
    ' 189 = "-"
    ' 33 = PageUp
    ' 34 = PageDown
    '
    Dim strYear As String
    Dim strMonth As String
    Dim strDay As String
    '
    If txtDate.Text = "" Then
        txtDate.Text = Format(Now, "d/m/yyyy")
        Exit Sub
    End If
    '
    strYear = Format(txtDate.Text, "yyyy")
    strMonth = Format(txtDate.Text, "mm")
    strDay = Format(txtDate.Text, "dd")
    '
    Select Case KeyCode
        Case 107, 187 ' добавляет один день
            txtDate.Text = Format(DateSerial(strYear, strMonth, strDay) + _
                           1, "d/m/yyyy")
        Case 109, 189 ' убавляет на один день
            txtDate.Text = Format(DateSerial(strYear, strMonth, strDay) - _
                           1, "d/m/yyyy")
        Case 33 ' увеличивает на один месяц
            txtDate.Text = Format(DateSerial(strYear, strMonth + 1, _
                           strDay), "d/m/yyyy")
        Case 34 ' уменьшает на один месяц
            txtDate.Text = Format(DateSerial(strYear, strMonth - 1, _
                          strDay), "d/m/yyyy")
    End Select
    '
End Sub
```

Этот способ коррекции даты очень полезен, так как гарантирует правильный формат даты в окне. Именно для того, чтобы избежать возможных ошибок при вводе, мы, установив свойство Locked равным True, заблокировали возможность редактировать в текстовом окне дату в явном виде.

Совет 129. Различайте термины KeyCode и KeyAscii

Обратите внимание, что при вводе данных в событие KeyPress передается код символа ASCII (KeyAscii), а для событий KeyDown/KeyUp — код клавиши (KeyCode).

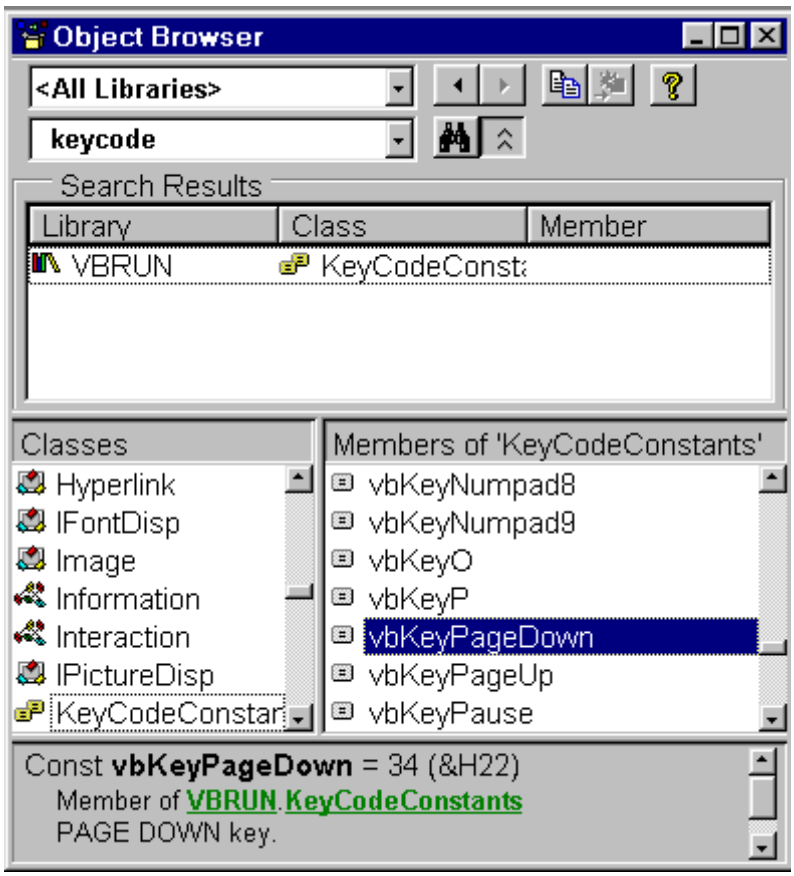
В первом случае мы получаем значимые символы в однобайтовой кодировке ANSI (событие KeyPress не реагирует на нажатие функциональных клавиш, клавиш редактирования, перемещения и пр., которые часто кодируются в двухбайтовой ANSI-кодировке). Нажатию одной и той же клавиши может соответствовать до четырех разных символов и их значений ASCII, например — "S s Ы ы". Код передается в виде ссылки (By Reference). Так что если вы измените его в этой процедуре, то объект (элемент управления) будет иметь дело с вашим значением. Например, такой код:

```
Sub Text1_KeyPress (KeyAscii As Interger)
   If KeyAscii = Asc("1") Then KeyAscii = Asc("2")
End If
```

будет вводить в текстовое окно двойку вместо единицы.

Во втором случае в процедуру передается код, закрепленный за данной клавишей (без учета текущей раскладки клавиатуры и регистра). При этом события KeyDown/KeyUp реагируют на нажатие (или освобождение) ЛЮБОЙ клавиши клавиатуры. Но нужно иметь в виду, что коды клавиш малой цифровой клавиатуры идентичны кодам соответствующих клавиш редактирования и перемещения. Не очень понятно, почему, но эти коды клавиш отличаются от значений SCAN-кодов клавиатуры.

В программе можно применять код клавиш в цифровом виде (как мы делали в предыдущем примере) или использовать встроенные константы VB. Например, клавиша PageDown описывается константой vbKeyPageDown=34. Список этих констант и их значений можно найти либо в справочной системе (для поиска нужного раздела введите во вкладке Index слово "KeyCode" для десятичного варианта значения или "Key Code" для шестнадцатеричного) либо через Object Browser:



Однако в списке этих констант почему-то отсутствуют некоторые клавиши — ('-=;',./). Впрочем, код любой клавиши вы можете легко определить с помощью такой простой программки:

```
Sub Text1_KeyDown (KeyCode As Interger, Shift As Integer)
    Debug.Print KeyCode
End If
```

Как видно из этого примера, в процедуры KeyDown/KeyUp передается также параметр Shift, значение которого отражает состояние клавиш Shift, Ctrl и Alt: при их нажатии устанавливается единица соответственно в 0-й, 1-й и 2-й разряд переменной. Например, комбинация клавиш Shift+Ctrl будет иметь значение 3 (двоичное 011).

Совет 130. Если хотите, то создавайте окна произвольной формы

В состав Win32 API входит одна очень занимательная функция, используя которую вы обнаружите, что в 32-разрядной версии Windows окно может иметь не только прямоугольную форму. На самом деле оно может быть любой конфигурации. Следующий пример демонстрирует работу с одной из так называемых функций региона (region functions) — SetWindowRgn.

Вначале введите такой код в раздел General Declarations:

```
Private Declare Function CreateEllipticRgn Lib "gdi32" _
  (ByVal X1 As Long, ByVal Y1 As Long, ByVal X2 As Long, _
  ByVal Y2 As Long) As Long

Private Declare Function SetWindowRgn Lib "user32" _
 (ByVal hWnd As Long, ByVal hRgn As Long, _
```

```
 ByVal bRedraw As Boolean) As Long
```

А затем напишите следующее для события Load:

```
Private Sub Form_Load()
   Show ' Вывод формы
   SetWindowRgn hWnd, CreateEllipticRgn(0, 0, 300, 200), True
End Sub
```

Вот и все. Запустите данный пример на выполнение, и VB выведет на экран форму в виде эллипса.

Совет 131. Управление размещением на экране элементов среды в VB5

После установки VB вы можете обнаружить, что инструментальные окна среды как бы "подключены" к основному окну и вы лишены возможности свободно управлять их размерами и размещением на экране. (В VB5 используется 9 таких окон: Immediate Window, Locals Window, Watches Window, Project Explorer, Properties Window, Object Browser, Form Layout, Toolbox и Color Palette.) Это означает, что для них установлен режим Dockable, снять который можно во вкладке Docking диалогового окна Options (меню Tools команда Options). Причем установка/снятие режима Dockable осуществляется отдельно для каждого окна (их список приведен во вкладке Docking).

Совет 132. Не используйте звуковой сигнал при нажатии клавиши Enter

Введя какую-либо информацию в текстовое окно, а затем нажав клавишу Enter, вы услышите звуковой сигнал. Чтобы избежать этого, запишите в событие KeyPress следующий код:

```
If KeyAscii = Asc(vbCr) Then KeyAscii = 0
```

Андрей Колесов, Ольга Павлова

© 1998, Андрей Колесов, Ольга Павлова
Авторский вариант. Статья была опубликована с незначительной литературной правкой в журнале "КомпьютерПресс" N 8/98, с.146-152.

Совет 133. Управление длиной элемента списка ComboBox, вариант 2

(совет прислал Михайлов Александр)

В КомпьютерПресс N 3'98 был опубликован Совет 126[7], где описывалась конструкция, способная компенсировать отсутствующее у элемента управления ComboBox свойство MaxLength. Предлагаю еще один способ реализации этого свойства с помощью функции API — SendMessage.

При работе с Win32 API (VB4/32 и VB5) необходимо описать данную функцию и константу CB_LIMITTEXT (как ее параметр) в таком виде:

[7] http://www.visual.2000.ru/develop/ms-vb/tips/9803.htm

```
Private Declare Function SendMessage Lib _
  "user32" Alias SendMessageA" (ByVal hWnd _
  As Long, ByVal wMsg As Long, ByVal wParam _
  As Long, lParam As Any) As Long
Private Const CB_LIMITTEXT = &H141
```

Для Win16 API (VB4/16 и более ранние версии VB) описание выглядит следующим образом:

```
Private Declare Function SendMessage Lib _
  "User" (ByVal hWnd As Integer, ByVal wMsg _
  As Integer, ByVal wParam As Integer, _
  lParam As Any) As Long
Private Const WM_USER = &H400
Private Const CB_LIMITTEXT = (WM_USER+1)
```

(Все описания функций API и констант взяты из файлов WIN31API.TXT и WIN32API.TXT, поставляемых с Visual Basic 4.)

Для установки ограничений ввода для списков, находящихся в форме, достаточно в событии Load вызвать функцию SendMessage для каждого элемента управления ComboBox с соответствующим параметром MaxLengthN — максимальным количеством символов, которое можно напечатать в поле ввода элемента ComboBox.

```
Sub Form_Load()
  Call SendMessage(Combo1.hWnd, CB_LIMITTEXT, MaxLength1, 0)
  Call SendMessage(Combo2.hWnd, CB_LIMITTEXT, MaxLength2, 0)
. . .
  Call SendMessage(ComboN.hWnd, CB_LIMITTEXT, MaxLengthN, 0)
End Sub
```

Для Win32 передаваемые параметры имеют тип Long, а для Win16 — Integer. Обратите внимание, что, хотя здесь используются процедуры-функции, к ним можно обращаться с помощью оператора Call. В этом случае просто не производится передача в вызывающую программу возвращаемого значения функции.

Дополнительные замечания авторов рубрики

Как видим, VB с помощью функций API может производить установку параметров элементов управления методом, который по-английски называется "send messages to controls". Более того, такую установку можно проводить и для тех параметров, которые в явном виде не представлены свойствами элементов управления. Обратите внимание, что каждый программный объект (формы, элементы управления) имеет в среде Windows свой уникальный номер-идентификатор, который хранится в свойстве hWnd.

Существуют два варианта передачи параметров объектам Windows: с помощью функций SendMessage и PostMessage. При работе Windows не всегда можно сразу установить свойства объекта из-за параллельности работы заданий. Функция SendMessage производит немедленный вызов соответствующей операции для указанного объекта и возвращает управление в вызывающую программу только после реального выполнения нужной установки. PostMessage устанавливает соответствующий запрос в очередь обращений к объекту и возвращает управление, не дожидаясь его выполнения.

Обе функции имеют целый ряд разновидностей. Например, SendMessageTimeout возвращает управление либо после выполнения установки, либо по истечении заданного

промежутка времени. А функция BroadcastSystemMessage делает установку параметров всех объектов или приложений указанного типа.

Список констант, задающих выполнение конкретных операций для этих функций, весьма обширен и состоит из двух больших наборов. Первый содержит константы, закрепленные для разных типов элементов управления (LB — List Box, CB — Combo Box, EM — Edit control и пр.). Во втором наборе используются константы, имена которых имеют вид WM_xxx, — они работают с несколькими типами объектов. Например, WM_COPY=&H301 выполняет копирование выделенного текста элементов управления Text Box или Combo Box в буфер обмена.

Однако нужно отметить, что довольно многие из таких API-операций не доступны для VB. (Например, WM_ENABLE=&HA, которая делает объект доступным или недоступным.) С другой стороны, ряд API-операций может быть реализован собственными средствами VB. (В частности, приведенное выше копирование выделенного текста — Clipboard.SetText Text1.SelText.)

Рекомендуем книгу по Win32 API

Полное описание функций API приведено в наборе для разработчиков Win32 SDK. Однако тем, кто занимается программированием на VB, мы очень рекомендуем книгу известного американского автора и разработчика VB-продуктов Дэна Эпплмана: "Dan Appleman's VB 5.0 Programmer's Guide to the Win32 API": Daniel Appleman, Macmillan Computer Publishing/Ziff-Davis Press, 1997. ISBN: 1-56276-446-2, 1548 с. (с компакт-диском). По единодушному мнению американских специалистов, с которым согласны и авторы рубрики, эта книга является самым надежным и полным пособием по данному вопросу. Так же, как и более ранняя книга Эпплмана, посвященная Win16 API, она сразу стала одним из бестселлеров.

В данной книге расширен материал первого издания, появившегося в 1995 г., — в ней отражены новые возможности VB 5.0. Первая часть книги посвящена общим принципам построения набора Windows API и совместной работе VB-программ с процедурами статических и динамических библиотек. В следующих двух частях приведены подробные справочные сведения по функциям API с учетом специфики VB, в том числе версий 4.0 и 5.0 (при этом указаны и те функции, которые не поддерживаются VB, что позволяет, в частности, составить лучшее представление об узких местах данной системы). В четвертой части и шести приложениях даны примеры их практического применения и различные дополнительные сведения.

Прилагаемый компакт-диск содержит ПОЛНЫЙ текст книги и всех программных примеров, а также три дополнительные главы ("Serial Communications", "Network Functions" и "API Types Libraries"), которые не вошли в печатный вариант. Кроме того, на диске находится ряд отдельных статей автора, большое число вспомогательных программ и демо-версии некоторых дополнительных продуктов.

Стоимость книги в США — 60 долл. Нам ее привез из Америки приятель, который предупредил перед отъездом, что у него не будет времени заниматься поиском книги. Однако потом он сообщил, что нашел ее в первом же книжном магазине небольшого провинциального городка.

Совет 134. Передача адреса процедуры в функцию API с помощью AddressOf

Один из традиционных недостатков систем MS Basic, в том числе и VB — отсутствие возможности работать напрямую с адресами функций (function pointer). Хотелось бы подчеркнуть, что это не является неизбежной характеристикой языка-интерпретатора (как это часто представляется некоторым программистам) или Basic, а связано с представлением о необходимой функциональности инструментария со стороны его создателя, Microsoft. В версии 5.0 (что никак не связано с появлением компилятора) Microsoft решила частично устранить данное узкое место в VB, позволив использовать с помощью оператора AddressOf передачу адреса функции в качестве аргумента при обращении к процедуре. Однако, что весьма примечательно, в состав VBA 5.0 по прихоти Microsoft этот оператор не попал.

Такое нововведение существенно упрощает использование в VB функций API (и аналогичных процедур в любых DLL), при выполнении которых, в свою очередь, осуществляется обратный вызов (callback, как это называется в C) к указанной пользовательской функции, написанной на VB. Мы говорим "упрощает", поскольку при большом желании и наличии соответствующих знаний применять подобные функции можно было и в более ранних версиях VB. Обычно для этого приходилось использовать некоторые нестандартные элементы управления. Работа с одним из них, DWCBK32D.OCX, описана в книге Эпплмана, а на компакт-диске записан сам модуль.

При использовании оператора AddressOf следует иметь в виду два ключевых момента:

- Его можно применять только в модуле кода (BAS) и только как часть аргумента процедуры. В операторе Declare этот аргумент определен как тип Any или Long. Передача адресов процедур, входящих в состав объекта или формы, не допускается. Сама процедура, на которую ссылается оператор AddressOf, должна находиться в том же проекте.
- С помощью AddressOf можно передавать адрес функции и в Basic-процедуры. Однако использовать его по прямому назначению (обращение к функции) там нельзя. В частности, модуль класса, написанный на VB, не может применять этот адрес для обратного обращения к своему контроллеру. Таким образом, к сожалению, VB по-прежнему не позволяет написать, например, простую универсальную подпрограмму для вычисления определенного интеграла для произвольной внешней подынтегральной функции. При передаче адреса функции через Basic-процедуры соответствующий параметр должен быть представлен переменной типа Long.

Использование оператора AddressOf требует правильного понимания идеи применения функций с обратным вызовом и четкого представления о том, как с ними работают конкретные DLL-функции. Отладка таких обращений довольно затруднительна, так как программа работает в том же самом процессе, что и среда разработки. В ряде случаев традиционные методы отладки здесь просто невозможны. Следует иметь в виду, что обработку возможных ошибок в вызываемой процедуре нужно выполнять именно в ней, поставив, например, в ее начало оператор On Error Resume Next.

Вы можете создать прототип функции обратного вызова в библиотеке DLL, созданной с помощью VC++ или другого аналогичного инструмента. Однако, чтобы применять AddressOf, этот прототип должен использовать при вызове преобразование __stdcall, а не __cdecl, которое реализуется по умолчанию.

Ниже приведен пример применения оператора AddressOf, взятый нами из книги "MS VB 5.0. Мастерская разработчика" Дж. Крейга и Дж. Уэбба. В нем производится обращение к API-функции EnumChildWindows, которая в свою очередь вызывает VB- функцию ChildWindowProc. Обратите внимание, что последняя должна находиться в модуле кода, а не формы. После запуска этого примера (установите Sub Main в поле Startup Object окна свойств проекта) в окно Immediate будут выведены все описатели (handles) окон среды разработки. Однако заметьте: если вы скомпилируете программы и запустите автономный EXE-модуль, список описателей будет совсем другим.

```
Option Explicit
Private Declare Function GetActiveWindow Lib "User32" () As Long

Private Declare Function EnumChildWindows _
  Lib "User32" ( ByVal hWnd As Long, ByVal _
  lpWndProc As Long, ByVal lp As Long) As Long

Sub Main()
  Dim hWnd As Long
  Dim x As Long
  ' получаем описатель активного окна
  hWnd = GetActiveWindow()
  If (hWnd) Then
    ' вызываем API-функцию EnumChildWindows,
    ' а та вызывает ChildWindowProc для каждого
    ' дочернего окна
    x = EnumChildWindows(hWnd, AddressOf _
    ChildWindowProc, 0)
  End If
End Sub

' вызывается API-функцией EnumChildWindows
Function ChildWindowProc(ByVal hWnd As Long, _
  ByVal lp As Long ) As Long
  ' параметры hWnd и lp передаются функцией
  ' EnumChildWindows
  Debug.Print "Window: "; hWnd
  ' сообщаем, успешен ли вызов
  ' (в стиле C, 1 - True, 0 - False)
  ChildWindowProc = 1
End Function
```

Совет 135. Как ускорить создание строковых буферов

Мы уже несколько раз обращали ваше внимание на необходимость внимательного отношения к используемым программным конструкциям. Осуществляя, казалось бы, одинаковые операции, они используют различные внутренние алгоритмы их реализации, весьма отличные по оптимальности. Особое внимание надо уделять работе со строковыми переменными — данные операции требуют существенно больше времени, чем численные. Причина состоит в том, что любая простая строковая операция присвоения требует динамического перераспределения памяти. Это усугубляется тем, что скорость данной операции существенно зависит от размера строковой переменной.

Рассмотрим пример случая, когда нужно сформировать буфер из большого количества данных в виде строковой переменной. В листинге 135[8] приведена процедура BuildBuffer, с

[8] http://www.visual.2000.ru/develop/ms-vb/tips/lst135.htm

помощью которой были протестированы три варианта решения этой задачи (результаты приведены ниже):

1. Первый вариант реализует наиболее очевидный и простой алгоритм:

```
2.   strBuffer = strBuffer & strNewData
```

   Однако с ростом числа циклов lCount время операций резко возрастает — почти в квадратичной зависимости (что является следствием увеличения размера буфера).

3. Во втором варианте используется промежуточный, временный буфер. Когда он становится достаточно большим, его содержимое добавляется к основному буферу, затем содержимое временного буфера очищается и работа продолжается. Скорость операций возрастает в десятки и даже сотни раз. Однако с какого-то момента и здесь рост идет по нелинейному закону.
4. Третий вариант реализует принципиально другой механизм. Ключевым моментом здесь является использование операции Mid$ вместо сложения двух переменных. Если в предыдущем случае эффект достигается за счет уменьшения размера накапливающего буфера, то здесь сводятся к минимуму операции динамического резервирования памяти — Mid$ записывает строку в тот же самый буфер достаточно большой длины. В результате скорость операций не просто увеличивается: время их выполнения практически не зависит от размера буфера и растет строго пропорционально числу циклов.

   Основная проблема здесь — контроль и автоматическое увеличение этого буфера (второй вопрос решается очень просто — методом удвоения буфера — начиная с длины, равной одному байту!). Обратите внимание, что вычисление длины добавляемой строки специально производится внутри цикла, чтобы показать, что она может быть различной. Конечно, сложность программного кода возрастает, но посмотрите, как растет быстродействие! В ряде случаев это бывает весьма критичным.

Таблица. Результаты тестирования программы из Совета 135

Количество циклов, lCount Время, сек.	**Вариант 1**	**Вариант 2**	**Вариант 3**
1000	1	-	-
2000	5	-	-
3000	10	-	-
4000	21	1	-
7000	61	-	-
10000	131	3	-
30000	-	15	0
50000	-	36	1

100000	-	139	2
200000	-	-	5

Примечание. *Выполнение в среде VB 5.0, процессор Pentium/MMX 200 МГц, 32 Мбайт ОЗУ; "-" замер не проводился.*

Совет 136. Управление вводом в текстовых полях

При создании пользовательских интерфейсов достаточно часто встает задача управления вводом данных в текстовых полях окон, например в элементе управления Text Box. Это может быть связано, например, с блокировкой ввода определенных символов или их автоматической заменой (мы уже давали ряд советов на данную тему). Такое управление реализуется с помощью написания соответствующего программного кода в событийной процедуре KeyPress (по поводу отличия кодов KeyAscii и KeyCode см. Совет 129[9]).

Однако при разработке приложений бывает полезно использовать некую единую универсальную процедуру для выполнения таких преобразований. Выгоды здесь две: возможна некоторая экономия объема программы, а самое главное, обеспечивается гибкое управление режимами ввода непосредственно в процессе работы приложения. Один из вариантов такой программы приведен в листинге 136[10]. Для ее использования достаточно, указав нужное значение Filter$, вставить в процедуры KeyPress следующее обращение:

```
Call ChangeKeyAscii(KeyAscii, Filter$)
```

Вот некоторые примеры значений параметра Filter$:

- "" — отсутствие преобразования;
- "*" — блокировка ввода двойных кавычек KeyAscii = 34 (частая проблема, когда вводят название с бумажного документа, например, шахта "Северная");
- "/" + Chr$(&H21) — блокировка ввода русских букв;
- "/" + Chr$(&H00) — блокировка ввода латинских букв с одновременной заменой строчных букв прописными;
- "/ 0123456789" — разрешен ввод только цифр;
- "*/0123456789" — запрещен ввод цифр, а также двойных кавычек.

В этом примере мы специально не использовали операторы Lcase/Ucase, чтобы показать возможность их альтернативной реализации (см. Советы 89 и 90[11]). Обратите внимание, что наш вариант будет работать для русских букв, даже если не установлена русская кодовая таблица cp1251. Разумеется, данный пример — лишь один из возможных вариантов таких процедур преобразования. Учитывая специфику своих приложений, вы легко создадите свои собственные вспомогательные подпрограммы.

[9] http://www.visual.2000.ru/develop/ms-vb/tips/9806.htm
[10] http://www.visual.2000.ru/develop/ms-vb/tips/lst136.htm
[11] http://www.visual.2000.ru/develop/ms-vb/tips/9703.htm

Совет 137. Как перетащить элементы из одного списка в другой

Чтобы взять элементы из одного списка и поместить их в другой, создайте два списка (lstDraggedItems и lstDroppedItems) и текстовое поле (txtItem) в форме (frmTip). Поместите такой код в событие Load для формы:

```
Private Sub Form_Load()
  ' Установите свойство Visible
  ' для текстового поля как False
  txtItem.Visible = False
  ' Добавьте элементы к списку 1 (lstDraggedItems)
  lstDraggedItems.AddItem "Яблоко"
  lstDraggedItems.AddItem "Апельсин"
  lstDraggedItems.AddItem "Грейпфрут"
  lstDraggedItems.AddItem "Банан"
  lstDraggedItems.AddItem "Лимон"
End Sub
```

В событии MouseDown списка lstDraggedItems напишите следующее:

```
Private Sub lstDraggedItems_MouseDown _
  (Button As Integer, Shift As Integer, _
  X As Single, Y As Single)
  '
  txtItem.Text = lstDraggedItems.Text
  txtItem.Top = Y + lstDraggedItems.Top
  txtItem.Left = X + lstDraggedItems.Left
  txtItem.Drag
End Sub
```

А в событии DragDrop списка lstDroppedItems введите такой код:

```
Private Sub lstDroppedItems_DragDrop _
  (Source As Control, X As Single, Y As Single)
  '
  If lstDraggedItems.ItemData _
    (lstDraggedItems.ListIndex) = 9 Then Exit Sub
  ' Убедимся, что данный элемент
  ' не будет перемещен снова
  lstDraggedItems.ItemData _
  (lstDraggedItems.ListIndex) = 9
  lstDroppedItems.AddItem txtItem.Text
End Sub
```

Теперь запустите этот тест на выполнение и попробуйте перетащить элементы из списка lstDraggedItems и поместить их в список LstDroppedItems.

Обратите внимание, что вы не можете перемещать элементы второго списка в первый. Кроме того, перетаскиваемые элементы сохраняются в первом списке. Надеемся, вы сумеете легко устранить данные ограничения.

Совет 138. Используйте ключевое слово ParamArray для передачи произвольного числа параметров

Вы можете использовать ключевое слово ParamArray в строке описания метода, чтобы создать подпрограмму или функцию, которая принимает произвольное число параметров в процессе выполнения. Например, можно создать метод, который будет заполнять окно

списка некоторым количеством элементов, даже если вы не знаете, сколько элементов будет передаваться. Добавьте приведенный ниже метод к форме:

```
Public Sub FillList(ListControl As ListBox, ParamArray Items())
  Dim i As Variant
  With ListControl
    .Clear
    For Each i In Items
      .AddItem i
    Next
  End With
End Sub
```

Обратите внимание, что ключевое слово ParamArray идет ВПЕРЕДИ параметра в строке описания.

Используя эту процедуру (довольно универсальную!), можно заменить одним оператором сразу пять строк в процедуре Form_Load:

```
Call FillList (lstDraggedItems, "Яблоко", "Апельсин", _
"Грейпфрут", "Банан", "Лимон")
```

Совет 139. Создание нового контекстного меню

Следующая программа позволит вам заменить исходное контекстное меню на свое собственное. Для этого добавьте следующий код к форме или BAS-модулю:

```
Private Const WM_RBUTTONDOWN = &H204
Private Declare Function SendMessage Lib "user32" _
  Alias "SendMessageA" (ByVal hwnd As Long, ByVal _
  wMsg As Long, ByVal wParam As Long, lParam As Any) As Long

Public Sub OpenContextMenu(FormName As Form, MenuName As Menu)
'
  ' Говорит системе, что пользователь щелкнул
  ' правой кнопкой мыши на форме
  Call SendMessage(FormName.hwnd, _
  WM_RBUTTONDOWN, 0, 0&)
  ' Показывает контекстное меню
  FormName.PopupMenu MenuName
End Sub
```

После этого с помощью редактора Visual Basic Menu Editor и приведенной здесь таблицы создайте простое меню.

Caption	**Name**	**Visible**
Контекстное меню	mnuContext	No
Первый элемент	mnuContext1	
Второй элемент	mnuContext2	

Обратите внимание, что два последних элемента смещены на один уровень (...) и что у первого элемента ("Контекстное меню") свойство Visible установлено как NO.

Теперь добавьте текстовое поле к форме и введите следующий код в событие MouseDown для этого элемента управления:

```
Private Sub Text1_MouseDown(Button As Integer, _
  Shift As Integer, X As Single, Y As Single)
  '
  If Button = vbRightButton Then
    Call OpenContextMenu(Me, Me.mnuContext)
  End If
End Sub
```

Совет 140. Используйте функцию FreeFile для предотвращения конфликтов при открытии файлов

VB и Access позволяют жестко кодировать номера файлов при использовании оператора File Open. Например:

```
Open "myfile.txt" For Append As #1
Print #1,"строка текста"
Close #1
```

Проблема здесь состоит в том, что вы никогда не знаете, какие номера файлов могут применяться где-нибудь еще в вашей программе. Если вы попытаетесь открыть файл с номером, который уже занят, то получите сообщение об ошибке. Чтобы избежать такой ситуации, следует всегда использовать функцию FreeFile, которая возвращает следующий по порядку свободный номер файла. Например:

```
intFile=FreeFile
Open "myfile.txt" For Append As #intFile
Print #intFile,"строка текста"
Close #intFile
```

Здесь следует иметь в виду следующее:

- номер файла фиксируется как занятый в момент его открытия. Поэтому определять очередной свободный номер следует непосредственно перед его открытием (как это сделано в примере);
- при закрытии файла номер освобождается;
- функция выборки свободного номера файла имеет вид:
- `FreeFile [(argument)],`

  где аргумент определяет диапазон возможных номеров: 0 (по умолчанию) — от 1 до 255, 1 — от 256 до 511. Документация по VB рекомендует использовать первый диапазон для файлов, недоступных для внешних приложений, а второй — для доступных.

Совет 141. Как предупредить пользователей о разрешении экрана

Здесь приведен способ предупреждения пользователя о том, что разрешение экрана его компьютера не подходит для выполнения приложения. Вначале создайте функцию с именем CheckRez:

```
Public Function CheckRez(pixelWidth As Long, _
  pixelHeight As Long) As Boolean
  '
  Dim lngTwipsX As Long
  Dim lngTwipsY As Long
  '
  ' Преобразование пикселов
  lngTwipsX = pixelWidth * 15
```

```
  lngTwipsY = pixelHeight * 15
  '
  ' Проверка текущих параметров
  If lngTwipsX <> Screen.Width Then
    CheckRez = False
  Else
    If lngTwipsY <> Screen.Height Then
      CheckRez = False
   Else
     CheckRez = True
   End If
  End If
End Function
```

После этого поместите следующий код в начало программы:

```
If CheckRez(640, 480) = False Then
  MsgBox "Неправильный размер экрана!"
Else
  MsgBox "Разрешение экрана соответствует!"
End If
```

Совет 142. Простой способ определения логической переменной

Довольно часто требуется зафиксировать некоторое соотношение чисел с помощью логической переменной. Пример этого приведен в предыдущем совете в следующем фрагменте:

```
If lngTwipsY <> Screen.Height Then
  CheckRez = False
Else
  CheckRez = True
End If
```

Однако гораздо проще записать его так:

```
CheckRez = (lngTwipsY = Screen.Height)
```

Совет 143. Как быстро выделить текст для события GotFocus

При вводе данных часто бывает необходимо выделить текущее значение элемента управления, когда в него передается фокус. Это позволяет пользователю немедленно начать ввод данных поверх предыдущего значения. Ниже приводится небольшая подпрограмма, которая предназначена именно для такой цели:

```
Public Sub FocusMe(ctlName As Control)
'
  With ctlName
   .SelStart = 0
   .SelLength = Len(ctlName)
  End With
End Sub
```

А теперь добавьте вызов к этой подпрограмме в событии GotFocus для тех элементов управления, которые используются при вводе данных:

```
Private Sub txtFocusMe_GotFocus()
```

```
  Call FocusMe(txtFocusMe)
End Sub
```

Андрей Колесов, Ольга Павлова

© 1998, Андрей Колесов, Ольга Павлова
Авторский вариант. Статья была опубликована с незначительной журнале "КомпьютерПресс" N 09/98, с. 160-164.

Совет 144a. Как принудительно изменить атрибут файла

Читатель присал нам такой вопрос:

Мне нужно после создания файла принудительно изменить его атрибут - дату создания - на предопределенный заранее. Полагаю, что правильно было бы использовать вызов API-функции SetFileTime. Я пытаюсь это сделать так (в этом примере мы убрали вспомогательный код программы с описанием переменных и процедур. - Прим. авт.):

```
Open FName For Output As #1
If SetFileTime(1, FileDate, FileDate, FileDate) <>0 Then ...
Close #1
```

Но что-то не получается...

Наш ответ:

В данном случае наш читатель допустил весьма характерную ошибку: смешал два варианта доступа к файлам - с помощью встроенных операторов VB и функций API. Действительно, возможности программиста при работе с файлами могут быть существенно расширены за счет использования соответствующего набора API-функций (как это ранее делалось в DOS с помощью функций DOS/BIOS).

Среди таких полезных операций можно упомянуть, например, возможность чтения за одно обращение больших массивов (а не по отдельным элементам), фиксацию состояния файлов без выполнения закрытия/открытия, коррекцию атрибутов и многое другое. При этом идентификация файлов выполняется с так называемыми описателями (handle). По смыслу они аналогичны понятию "логический номер" (ЛН), но вот нумерация их принципиально различается: для handle она ведется на уровне всей ОС, а для ЛН - отдельного приложения. Если сразу после запуска программы вы затребуете свободный номер, то для ЛН получите 1, а для handle это значение может быть что-то типа 49.

В данном случае после открытия файла с ЛН=1 было сделано обращение к функции SetFileTime для handle=1. При этом не понятно, к какому файлу он относится, и к тому же он вообще не был открыт в данной программе. Вывод таков: при работе с конкретным файлом можно пользоваться только одним типом доступа - либо операторами VB, либо API.

Примечание автора. *Уже после публикации этого совета мне попадалась информация о том, что можно получить номер описателя handle для файла, открытого с помощью ЛН. Поэтому, возможно, вполне реально использование одновременно двух режимов доступа. Нужно проверять...*

В качестве примера решения проблемы читателя предлагаем такой вариант:

```
' функции преобразования формата даты
Private Declare Function SystemTimeToFileTime& Lib _
  "kernel32" (lpSystemTIME As SYSTEMTIME, _
  lpFileTime As FILETIME)

Private Declare Function FileTimeToSystemTime& Lib _
  "kernel32" (lpFileTime As FILETIME, lpSystemTIME _
  As SYSTEMTIME)

' функции работы с файлами
Private Declare Function lopen& Lib "kernel32" _
  Alias "_lopen" (ByVal lpFileName As String, _
  ByVal wReadWhite As Long)

Private Declare Function lclose& Lib "kernel32" _
  Alias "_lclose" (ByVal hFile As Long)

Private Declare Function SetFileTime& Lib "kernel32" _
  (ByVal hFile As Long, lpCreationTime As FILETIME, _
  lpLastAccessTime As FILETIME, lpLastWriteTime As FILETIME)

' функция для анализа ошибок
Private Declare Function GetLastError& Lib "kernel32" ()

' для хранения даты во внутреннем формате
Private Type FILETIME
  dwLowDateTime As Long
  dwHighDateTime As Long
End Type

' для хранения даты в системном формате
Private Type SYSTEMTIME
  wYear As Integer
  wMonth As Integer
  wDayOfWeek As Integer
  wDay As Integer
  wHour As Integer
  wMinute As Integer
  wSecond As Integer
  wMilliseconds As Integer
End Type

Private Sub Form_Load()
  Dim SysTime As SYSTEMTIME, NowTime As FILETIME
  Dim FileName$, handleF&, wReadWrite&, k&, k1&
  ' дата в системном формате
  SysTime.wYear = 1997
  SysTime.wMonth = 10
  SysTime.wDay = 3
  ' преобразование даты во внутренний двоичный формат
  k& = SystemTimeToFileTime(SysTime, NowTime)
  '
  ' имя файла - он должен существовать
  FileName$ = "d:\d.txt"
  ' Работа с файлами только средствами функций API
  ' ВНИМАНИЕ! Для изменения атрибутов файла,
  ' он должен  быть открыт в режиме "разрешения
  ' записи", например:  режим "чтение-запись"
  ' Const OF_READWRITE& = 2
  wReadWrite& = 2
  ' Открытие файла
```

```
  handleF& = lopen&(FileName$, wReadWrite&)
  ' запись новых атрибутов даты
  k& = SetFileTime&(handleF&, NowTime, NowTime, NowTime)
  ' была ли ошибка? Можно проверить
  k1& = GetLastError ' код ошибки
  ' закрытие файла
  Call lclose(handleF&)
End Sub
```

Совет 144. Для тех, кто занимается геометрическими расчетами

Возможно, вам пригодятся две процедуры, которые приведены в модуле XY_TESTC.BAS (см. ниже). Они сохранились у нас еще со времен Basic/DOS, поэтому их текст и имеет такой вид (например, все ключевые слова записаны заглавными буквами). Процедура CircleTestXY определяет местоположение точки относительно фигуры-многоугольника (внутри или снаружи), CircleSquare вычисляет площадь многоугольника. Следует обратить внимание на то, что одна из вершин многоугольника задана в массиве дважды - в качестве начальной и конечной точки.

Кстати. *Раньше названия языков программирования и их ключевых слов было принято писать большими буквами. Однако в начале 90-х годов Международная Организация по Стандартам (ISO - International Standard Organization) приняла решение об изменении этого правила, С тех пор они пишутся так: первая буква - заглавная, остальные - прописные.*

```
DECLARE SUB CircleTestXY (xyd!(), Np%, x0!, y0!, kz%)
DECLARE SUB CircleSquare (xyd!(), Np%, Square!)
DEFINT I-N
'************************************************
'  Модуль XY_TESTC.BAS
'
' Процедуры:
' CircleTestXY - определение местоположения точки
' относительно фигуры-многоугольника
' CircleSquare - вычисление площади многоугольника
'
''''''
```

```
  ' x = xyd(1,i), y = xyd(2,i) ; i = 1 to Np
  '  (Np-1) - количество узлов
  '  координаты 1-й точки = координатам N-й
  '
  ' ВЫХОД: Square - площадь многоугольника

''''''
```

```
        IF kv = 2 OR kv = -2 THEN ' переход через две четверти
          IF x1 = x2 THEN kz = -100: EXIT FOR
          yb = (y2 * x1 - y1 * x2) / (x1 - x2)
          IF yb = 0 THEN kz = -100: EXIT FOR
          kv = kv * SGN(yb)
          IF kv1 = 2 OR kv1 = 4 THEN kv = -kv
        END IF
        kz = kz + kv
      END IF
    END IF
    x1 = x2: y1 = y2: kv1 = kv2
  NEXT
END SUB
```

Андрей Колесов, Ольга Павлова

© 1998, Андрей Колесов, Ольга Павлова
Авторский вариант. Статья была опубликована с незначительной литературной правкой в журнале "КомпьютерПресс" N 12/98, компакт-диск.

Совет 145. Определение положения указателя линейки прокрутки

Как известно, положение указателя (движка) линейки прокрутки характеризуется значением свойства Value, которое изменяется от значения Min до Max (свойства элемента управления, которые можно задать в окне Properties).

Сделайте такой небольшой опыт. Поместите на форме поле текста, а справа от него — вертикальную линейку прокрутки. Установите Min и Max равными 1 и 10 соответственно (по умолчанию они были 0 и 32767). В событие VScroll1_Change введите код:

```
Text1.Text = VScroll1.Value
```

Теперь нажмите F5 для запуска проекта. Прокручивая линейку, вы будете видеть в поле текста цифровое значение положения указателя. Обратите внимание, что свойства LargeChange и SmallChange элемента VScroll задают соответственно величину, на которую будет изменяться значение свойства Value данного элемента управления при щелчке пользователем области между движком и стрелкой прокрутки или при щелчке самой стрелки прокрутки.

Проблема же порой заключается в том, что минимальное значение Value соответствует верхнему положению указателя. В то же время в некоторых случаях бывает нужным, чтобы Value при перемещении указателя вниз не увеличивалось, а наоборот - уменьшалось. Сделать это можно просто, заменив программный код:

```
Text1.Text = VScroll1.Max - VScroll1.Value + VScroll1.Max
```

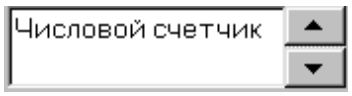
Однако еще проще сделать так: свойства Max и Min установите равными отрицательным значениям, например -1 и -10, а в событие Change запишите строку:

```
Text1.Text = Abs(VScroll1.Value)
```

Теперь в обоих случаях верхнее положение указателя будет соответствовать максимальному значению заданного диапазона данных.

Совет 146. Создание числового счетчика

Предыдущий совет поможет нам в создании элемента управления, представляющего собой числовой счетчик, внешний вид которого приведен здесь:



Как легко догадаться, на самом деле это два стоящих рядом элемента управления Text и VScroll с одинаковыми значениями свойств Top и Height, причем последний подобран таким образом, что указатель линейки прокрутки стал невидимым. Теперь, щелкая стрелки линейки, мы увидим автоматическое изменение числового значения с нужным нам шагом (SmallChange). Для практического использования такого счетчика следует также установить его начальное значение, например, при загрузке формы:

```
Sub Form_Load()
  VScroll1.Value = VScroll1.Max
  Call VScroll1_Change()
End Sub
```

Совет 147. Как измерить протяженность текста

Порой бывает полезным узнать физическую длину текстовой строки (размер в единицах длины с учетом, в частности, размера и типа шрифта) при выводе ее на экран. В принципе для этого можно воспользоваться функциями Win API, но есть еще более простой способ - использовать свойство AutoSize элемента управления Label. Для начала поместите метку (labMeasure) на форму и установите ее свойство AutoSize как True, а свойство Visible - как False. То есть ширина метки будет автоматически устанавливаться равной длине строки, но при этом сам текст метки будет невидимым. После этого напишите такую функцию:

```
Private Function TextExtent(txt$ As String) As Integer
  labMeasure.Caption = txt$
  TextExtent = labMeasure.Width
End Function
```

Обратите внимание, что возвращаемая величина (TextExtent) измеряется в твипах. Твип (twip) - это независимая от экрана единица измерения, используемая для обеспечения того, чтобы местонахождение и пропорции элементов экрана в приложении всегда были одинаковы на всех типах дисплеев. Твип - единица измерения экрана, равная 1/20 точки принтера. В 1 логическом сантиметре содержится приблизительно 567 твипов, а в 1 логическом дюйме - 1440 твипов. Логический сантиметр (дюйм) - это длина элемента экрана, равная 1 сантиметру (дюйму) при печати.

Теперь, если вы захотите выяснить протяженность какого-либо текста, просто вызовите данную функцию с текстовой строкой в качестве параметра. Проиллюстрируем это на следующем примере. Поместим на форму еще один элемент управления - поле текста. Мы хотим выяснить, какую он должен иметь ширину, чтобы целиком вмещать строковую переменную a$ (у текстового поля нет свойства AutoSize). Обратите внимание, что элементы управления Label и TextBox содержат разное количество символов и при равной ширине позволяют записывать различное число видимых символов. Поэтому мы добавляем два пробела к строке A$:

```
Private Sub Form_Load()
```

```
  txt1$ = "Исходный текст"
  Text1.Width = TextExtent(" " & txt1$)
  Text1.Text = txt1$
End Sub
```

Совет 148. Используйте пользовательские диалоговые окна при редактировании ячеек элемента управления DBGrid

Существует простой способ добавления различных пользовательских диалоговых окон ко всем ячейкам элемента управления DBGrid. Для этого вначале поместите на форму компоненты DBGrid и Data. Установите свойства DatabaseName и RecordSource элемента управления Data как, например, biblio.mdb и Publishers. Затем установите свойство DataSource элемента управления DBGrid как Data1 (компонент Data).

Теперь напишите следующий код:

```
Dim strDBGridCell As String
Private Sub DBGrid1_AfterColEdit(ByVal ColIndex As Integer)
'
  DBGrid1.Columns(ColIndex) = strDBGridCell
End Sub

Private Sub DBGrid1_BeforeColEdit(ByVal _
  ColIndex As Integer, ByVal KeyAscii As Integer, _
  Cancel As Integer)
'
  strDBGridCell = InputBox("Edit DBGrid Cell:", , _
  DBGrid1.Columns(ColIndex))
End Sub
```

Теперь, когда пользователь попытается отредактировать какую-либо ячейку элемента управления DBGrid, на экран будет выводиться запрос InputBox для осуществления ввода. Если хотите, то можете заменить этот запрос на любое другое пользовательское диалоговое окно.

Совет 149. При формировании констант типа Long используйте суффикс &

Уже довольно давно, в Совете 80[12] мы обращали ваше внимание на специфику работы с переменными типа Long (&) при их использовании в качестве масок в логических операциях.

Например, если выполнить такой фрагмент программы:

```
Code& = &H80040113
Result& = Code& And &HFFFF
```

то содержимое Result& будет равно не &H0113, как хотелось бы, а &H80040113.

Суть проблемы заключается в том, что переменные Integer и Long являются переменными со ЗНАКОМ, и нужно быть очень внимательным при переходе от беззнакового представления числа (&H...) к знаковому (цифровому десятичному). Рассмотрим такую конструкцию:

[12] http://www.visual.2000.ru/develop/ms-vb/tips/9702.htm

```
Const Mask& = &HFFFF
Print HEX$(Mask&)     ' будет напечатано &HFFFFFFFF !!!!
```

Дело в том, что константа &HFFFF сначала автоматически представляется в виде переменной Integer и в числовом выражении равна -1. А уже затем при присвоении Mask& = &HFFFF происходит преобразование из Integer в Long и переменная Mask& = -1 (&HFFFFFFFF)! Аналогичные преобразования происходят и в приведенной выше операции Result& = Code& And &HFFFF.

Тогда мы посоветовали читателям для определения констант типа Long, в том числе в арифметических или логических операциях, в диапазоне значений &H8000- &HFFFF (32768-65535) не использовать беззнаковое шестнадцатеричное представление, применяя только десятичное.

Так вот, мы были в общем-то не правы. На самом деле имеется возможность задания "чистых" констант типа Long независимо от диапазона их значений. Действительно, константа &HFFFF преобразуется в целочисленную переменную типа Integer. Но если справа к ней добавить еще один символ &, то сразу получится переменная Long. Попробуйте сами:

```
Const Mask& = &HFFFF&
Print HEX$(Mask&)
```

будет напечатано &HFFFF - то, что нам и нужно было получить. Причем машинный код этого фрагмента стал компактнее из-за отсутствия дополнительных внутренних преобразований данных.

Совет 150. Использование параметра Alias при работе с функциями API

Ряд функций Windows API содержит параметры, которые могут определяться различным образом. Например, при вызове функции WinHelp последний параметр может передавать данные типа Long или String в зависимости от заданного действия.

Visual Basic позволяет объявлять такой тип данных, как Any, при вызовах функций API, но это может привести к ошибкам несовпадения типов или даже к аварийному отказу системы, если значение имеет неправильный тип.

Поэтому во избежание подобных ошибок или для улучшения проверки типов данных во время выполнения приложения бывает полезным объявление нескольких версий одной и той же функции API. Написав для каждого возможного типа параметра свое объявление функции, вы осуществляете более жесткую проверку типов данных.

Для иллюстрации данной методики добавьте к модулю формы следующие функции API и константы. Обратите внимание, что два описания функции API отличаются друг от друга только своим названием (WinHelp и WinHelpSearch) и объявлением типа последнего параметра (dwData As Long и dwData As String).

```
' Объявления API-функции WinHelp
Private Declare Function WinHelp Lib "user32" Alias _
  "WinHelpA" (ByVal hwnd As Long, ByVal lpHelpFile _
  As String, ByVal wCommand As Long, ByVal dwData _
  As Long) As Long
'
Private Declare Function WinHelpSearch Lib "user32" _
```

```
  Alias "WinHelpA" (ByVal hwnd As Long, ByVal _
  lpHelpFile As String, ByVal wCommand As Long, _
  ByVal dwData As String) As Long
'
Private Const HELP_PARTIALKEY = &H105&
Private Const HELP_HELPONHELP = &H4&
Private Const HelpFile = _
"c:\program files\devstudio\vb5\help\vb5.hlp"
```

Примечание. *Мы учли наш предыдущий совет, указав суффикс & при определении констант. В данном случае (для данных конкретных значений) это было необязательно. Но, во-первых, мы подчеркнули, что формируются константы именно Long. А во-вторых, машинный код стал короче из-за отсутствия дополнительных преобразований.*

Теперь поместите на форму две командные кнопки (cmdHelpAbout и cmdHelpSearch) и напишите для них следующий код. Измените путь к файлу Справки в соответствии со своими установками для Visual Basic.

```
Private Sub cmdHelpAbout_Click()
'
  WinHelp Me.hwnd, HelpFile, HELP_HELPONHELP, &H0
End Sub

Private Sub cmdHelpSearch_Click()
'
  WinHelpSearch Me.hwnd, HelpFile, HELP_PARTIALKEY, "option"
End Sub
```

Запустите полученный проект на выполнение (F5). Если вы щелкните кнопку cmdHelpAbout, то увидите раздел Справки об использовании справочной системы. Если же вы щелкните кнопку cmdHelpSearch, то на экране появится список элементов Справки по теме option.

Совет 151. Как дать возможность пользователям отменить выгрузку форм

Существует множество различных способов выгрузить форму: щелкнуть кнопку Exit или соответствующую команду меню, щелкнуть кнопку со знаком X в верхнем правом углу формы, выбрать команду Close из всплывающего меню окна формы в верхнем левом углу. Можно даже произвести отмену выполнения программы из менеджера задач или перезагрузить компьютер.

Так или иначе, но было бы полезно дать пользователям возможность отменить операцию выгрузки формы, проводимую одним из вышеперечисленных способов. Для этого следует поместить код, осуществляющий проверку выгрузки формы, в событие QueryUnload для этой формы. Данное событие инициируется независимо от метода, используемого для выгрузки формы:

```
Private Sub Form_QueryUnload(Cancel As Integer, _
  UnloadMode As Integer)
  '
  ' универсальная проверка выгрузки формы
  Dim strQuestion As String
  Dim intAnswer As Integer
  Dim aryMode As Variant
  '
  aryMode = Array("vbFormControlMenu", _
```

```
    "vbFormCode", "vbAppWindows", _
    "vbAppTaskManager", "vbFormMDIForm")
  strQuestion = "Вы готовы выгрузить эту форму?"
  intAnswer = MsgBox(strQuestion, _
   vbQuestion + vbYesNo, aryMode(UnloadMode))
  If intAnswer = vbNo Then Cancel = -1
End Sub
```

Совет 152. Создание "автоматического" поля текста

С помощью приведенной здесь программы вы можете создать поле текста (TextBox), аналогичное тем, которые реализованы в последних версиях Microsoft Excel или Internet Explorer. Иными словами, каждый раз, когда вы будете вводить какой-либо текст в это поле, первые буквы строки будут сравниваться с элементами невидимого списка, а затем программа сама определит, как следует дописать данную строку, и сделает это за вас.

Для реализации такого поля текста вначале добавьте окно списка к своей форме и установите его свойство Visible как False. В событии Form_Load заполним это окно списка некоторыми элементами. В реальном приложении добавление новых элементов списка следует осуществлять каждый раз, когда пользователь закончит ввод строки. Поместите на форму элемент управления TextBox и введите следующий код:

```
Option Explicit
 #If Win32 Then  ' 32-разрядная версия VB
   Private Const LB_FINDSTRING = &H18F
   Private Declare Function SendMessage Lib _
     "User32" Alias "SendMessageA" (ByVal hWnd _
     As Long, ByVal  wMsg As Long, ByVal wParam _
     As Long, lParam As _ Any) As Long
 #Else  ' 16-разрядная версия VB
   Private Const WM_USER = &H400
   Private Const LB_FINDSTRING = (WM_USER + 16)
   Private Declare Function SendMessage Lib _
     "User" (ByVal hWnd As Integer, ByVal wMsg _
     As Integer,  ByVal wParam As Integer, lParam _
     As Any) As Long
 #End If

Private Sub Form_Load()
  List1.AddItem "Апельсин"
  List1.AddItem "Банан"
  List1.AddItem "Яблоко"
  List1.AddItem "Персик"
  List1.AddItem "Ананас"
  List1.AddItem "Авокадо"
End Sub

Private Sub Text1_Change()
  Dim pos As Long
  List1.ListIndex = SendMessage(List1.hWnd, _
    LB_FINDSTRING, -1, ByVal CStr(Text1.Text))
  If List1.ListIndex = -1 Then
    pos = Text1.SelStart
  Else
    pos = Text1.SelStart
    Text1.Text = List1
    Text1.SelStart = pos
    Text1.SelLength = Len(Text1.Text) - pos End If
End Sub
```

```
Private Sub Text1_KeyDown(KeyCode As Integer, Shift As Integer)
  '
  On Error Resume Next
  If KeyCode = 8 Then ' Backspace
    If Text1.SelLength <> 0 Then
      Text1.Text = Mid$(Text1, 1, Text1.SelStart - 1)
      KeyCode = 0
    End If
  ElseIf KeyCode = 46 Then ' Del
     If Text1.SelLength <> 0 And _
       Text1.SelStart <> 0 Then
       KeyCode = 0
    End If
  End If
End Sub
```

Теперь запустите созданный вами проект (F5) и попробуйте ввести какой-либо текст.

Совет 153. Управление кнопками Minimize/Maximize на MDI-форме

В отличие от других форм MDI-формы не имеют свойств MinButton и MaxButton, с помощью которых можно включать или отключать кнопки Minimize и Maximize на форме. Если добавить следующий код к событию Load родительской MDI-формы, то он отключит обе эти кнопки. Если же вы хотите отключить только одну из них, то поставьте знак комментария у соответствующей строки:

```
Private Sub MDIForm_Load()
  Dim lWnd As Long
  lWnd = GetWindowLong(Me.hWnd, GWL_STYLE)
  lWnd = lWnd And Not (WS_MINIMIZEBOX)
  lWnd = lWnd And Not (WS_MAXIMIZEBOX)
  lWnd = SetWindowLong(Me.hWnd, GWL_STYLE, lWnd)
End Sub
```

Затем введите следующий код (который включает объявления необходимых API-функций) в текст самого BAS-модуля:

```
Option Explicit
 ' описание API-функций и констант
 #If Win32 Then
   Private Declare Function SetWindowLong Lib _
     "user32" Alias "SetWindowLongA" (ByVal _
     hWnd As Long, ByVal nIndex As Long, ByVal _
     dwNewLong As Long) As Long
   Private Declare Function GetWindowLong Lib _
     "user32" Alias "GetWindowLongA" (ByVal _
    hWnd As Long, ByVal nIndex As Long) As Long
 #Else
   Declare Function SetWindowLong Lib "User" _
     (ByVal hWnd As Integer, ByVal nIndex As _
     Integer, ByVal dwNewLong As Long) As Long
   Declare Function GetWindowLong Lib "User" _
     (ByVal hWnd As Integer, ByVal nIndex As _
     Integer) As Long
 #End If

Const WS_MINIMIZEBOX = &H20000
Const WS_MAXIMIZEBOX = &H10000
Const GWL_STYLE = (-16)
```

Совет 154. Как сделать видимыми кнопки Minimize/Maximize

В некоторых случаях бывает необходимо задать свойство BorderStyle какой-либо формы как Fixed Dialog, и тогда VB не выводит кнопки Minimize и Maximize в поле заголовка формы. Теперь, если задать для этой же формы свойства MinButton и MaxButton как True, команды Minimize и Maximize в контекстном меню формы станут видимыми. Однако сами кнопки на форме останутся по-прежнему невидимыми. Чтобы исправить эту ошибку, введите описание API-функций и констант из предыдущего совета и запишите следующий код в стандартный модуль:

```
Public Sub SetCaptionButtons(Frm As Form)
  Dim lRet As Long
  lRet = GetWindowLong(Frm.hWnd, GWL_STYLE)
  SetWindowLong Frm.hWnd, GWL_STYLE, lRet Or _
  WS_MINIMIZEBOX * (Abs(Frm.MinButton)) Or _
  WS_MAXIMIZEBOX * (Abs(Frm.MaxButton))
End Sub
```

Теперь следует вызвать подпрограмму SetCaptionButtons из события Form_Load, передав ссылку на вашу форму.

Андрей Колесов, Ольга Павлова

© 1998, Андрей Колесов, Ольга Павлова
Авторский вариант. Первая часть статьи, опубликованой с незначительной литературной правкой в журнале "КомпьютерПресс" N 01/99, компакт-диск.

Совет 155. Используйте VB API Viewer

При работе с внешними функциями DLL-библиотек их описание с помощью оператора Declare должно быть включено в состав каждого программного компонента проекта (форма, модуль и пр.), где они используются. Однако при этом нужно быть особенно внимательным — ошибка может повлечь неприятности в виде зависания программы.

Кроме того, многие такие функции используют в качестве параметров специальные структуры данных Type, а для управления режимами выполнения функций — определенные значения параметров. Здесь нужно обратить внимание на следующую особенность современного стиля документации и литературы по VB: если в описании говорится об использовании параметра или типа данных, имя которых обозначено заглавными буквами (например LB_GETITEMHEIGHT), то имеются в виду некоторые стандартные значения констант или структур.

Для упрощения работы с наиболее часто используемыми внешними функциями — стандартным набором Windows — в состав VB 5.0 (и более ранних версий) входят два файла, содержащие описания процедур, а также константы и структуры данных: WIN32API.TXT (собственно набор Win32 API) и MAPI32.TXT (дополнительный набор Message API — почтовых функций). А для удобства работы с ними — утилита APILOAD.EXE и ее описание APILOAD.TXT. Все эти файлы находятся в подкаталоге \Winapi основного каталога Visual Basic. (Для установки файлов подкаталога \Winapi в программе Setup Visual Basic в разделе Online Help and Samples должен быть отмечен пункт Windows API Reference.)

Пользоваться информацией, содержащейся в файлах описаний, в принципе можно с помощью любого текстового редактора, копируя через буфер обмена необходимые данные в VB. Но удобнее для этого использовать стандартную программу APILOAD.EXE (VB API Viewer).

Предупреждение. Не следует загружать файл WIN32API.TXT в проект в виде программного модуля. Файл занимает слишком много места в памяти. Обычной практикой является копирование нужных описаний, констант и структур данных в соответствующие программные модули проекта.

Ее можно запускать из среды Windows, однако это лучше делать из среды VB 5.0, для чего нужно записать команду вызова APILOAD.EXE в меню Add-Ins. Для этого в среде VB меню Add-Ins выберите команду Add-In Manager, в появившемся диалоговом окне в списке Available Add-Ins установите флажок VB API Viewer (рис.1) и нажмите OK. Точно так же можно установить обращение и к другим дополнениям VB.

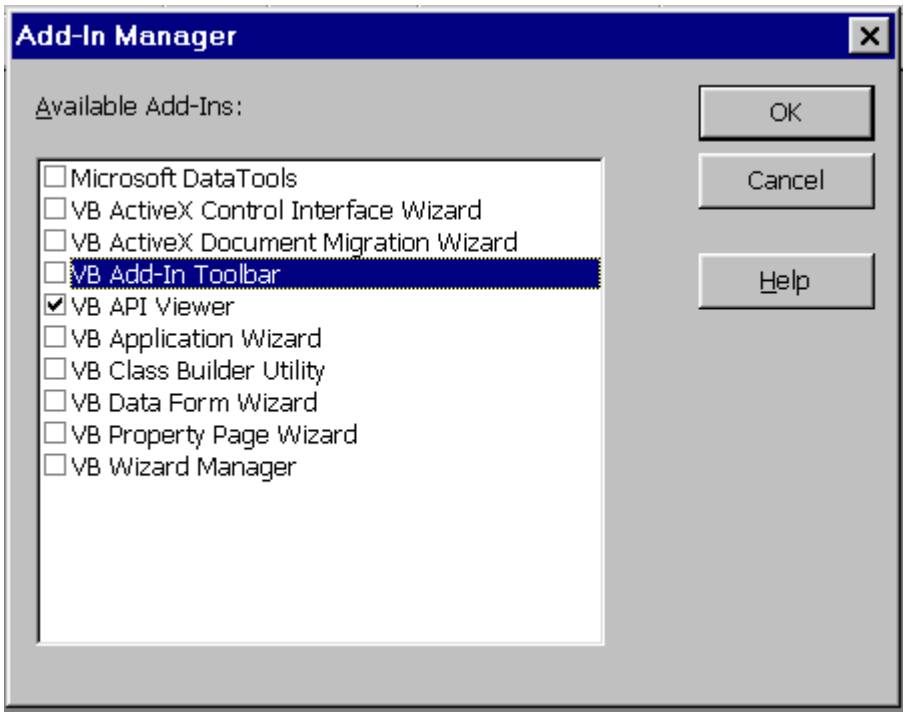


Рис. 1

Примечание. *Если в списке Available Add-Ins нет стандартного набора дополнений VB, то нужно запустить программу Setup VB и отметить раздел Wizards and Templates.*

Далее все происходит достаточно просто — когда вам нужно получить описания, константы и типы данных для работы с API, выберите команду Add-Ins|API Viewer, после чего появится окно утилиты (рис.2). Далее необходимо загрузить нужный файл описаний, например WIN32API.TXT, с помощью команды Load Text File из меню File. Потом, устанавливая в списке API Type нужный вид информации, можно просматривать содержимое файла в списке Available Items. Для быстрого поиска можно использовать кнопку Search.

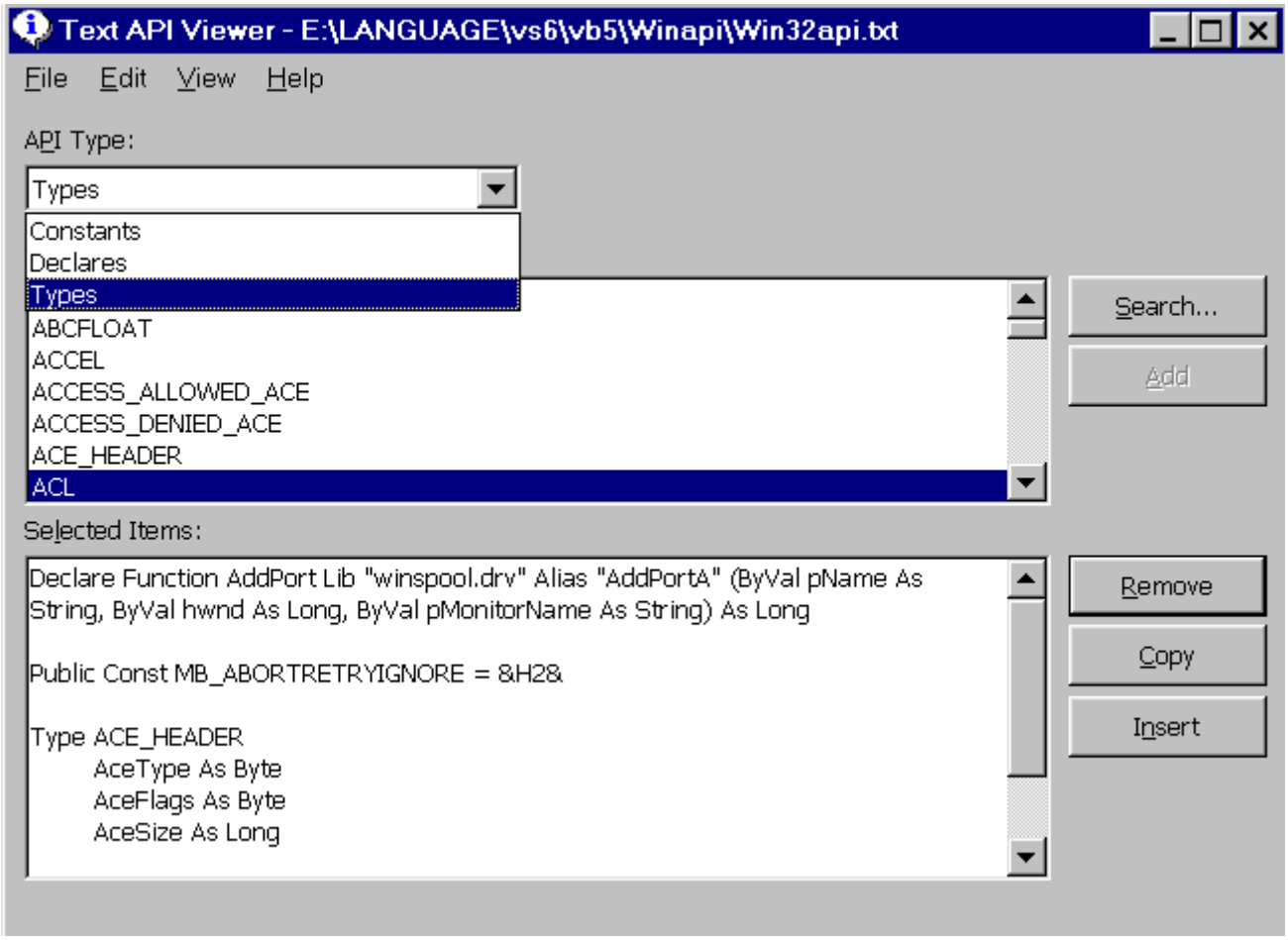


Рис. 2

Для более быстрой загрузки файла описания можно преобразовать тестовый файл в базу данных Jet (соответствующая команда находится в меню File). Но при работе с MDB-файлом почему-то пропадает кнопка Search.

Добавление описаний из исходного файла в проект VB выполняется следующим образом. В списке Available Items выделяется нужный элемент, а потом с помощью кнопки Add его содержимое переносится в окно Selected Items. Чтобы удалить информацию из окна Selected Items, надо установить на нее курсор и нажать кнопку Remove.

Скопировать содержимое окна Selected Items в проект VB можно двумя способами.

1. Нажав кнопку Copy, скопировать данные в буфер обмена, а потом, перейдя в среду VB, вставить информацию в нужное место.
2. Нажав кнопку Insert, сразу скопировать информацию в раздел Declarations активного в этот момент программного модуля VB. (Это самый удобный вариант — ведь обращение за описанием API-функции производится обычно как раз при активизации соответствующего модуля.)

ВНИМАНИЕ! ВСТРЕЧАЮТСЯ ОШИБКИ! *К сожалению, в файле описаний встречаются ошибки. В частности, в WIN32API.TXT приведено такое описание функции GetCurrentDirectory (о ней рассказывается в Совете 164[13]):*

[13] http://www.visual.2000.ru/develop/ms-vb/tips/9901-2.htm

```
Declare Function GetCurrentDirectory Lib "kernel32" _
Alias "GetCurrentDirectory"...
```

Здесь видно, что имя функции и ее альтернативное название совпадают. Соответственно при копировании этой строки в программный код редактор VB совершенно логично автоматически убирает команду Alias. При запуске программы на выполнение выдается сообщение об отсутствии функции в DLL- библиотеке. Ошибка очевидна — в конце названия функции после команды Alias пропала буква "A": ... Alias "GetCurrentDirectoryA"... Имейте это в виду, если вам встретится аналогичная ошибка в других описаниях.

А зачем вообще нужна команда Alias? Об этом мы расскажем в одном из следующих выпусков "Советов".

Совет 156. Автоматическое выделение элементов списка

Хотите сделать так, чтобы ваше окно списка вело себя точно так же, как меню? А именно: чтобы когда пользователь перемещал курсор по списку, то выделенный элемент перемещался бы вместе с курсором? Это вполне возможно.

Ключевым моментом в решении данной задачи является то, что каждый элемент стандартного окна списка имеет одинаковую высоту. И если вы знаете эту высоту (значение для самого верхнего видимого элемента) и Y-координату курсора, то вычисление элемента, находящегося под курсором, не составит никакого труда.

Воспользуемся API-функцией SendMessage, чтобы получить высоту каждого элемента списка в пикселах (единица физического разрешения экрана). Для этого нужно обратиться к функции (мы писали о ней более подробно в Совете 133[14]) с параметрами wMsg = LB_GETITEMHEIGHT (= &H1A1) и wParam = lParam = 0. Возвращаемая величина и будет высотой элемента списка.

Событие List1_MouseMove передает Y-координату курсора в твипах (единица длины — 1/1440 дюйма). Поэтому нужно преобразовать полученную ранее высоту элемента списка из пикселов в твипы с помощью метода ScaleY.

Теперь поделим передаваемое значение Y на преобразованную высоту элемента списка, затем добавим значение свойства TopIndex — и мы получим элемент, находящийся под курсором. И, наконец, последнее — проверьте, чтобы вычисленное значение свойства ListIndex не было равно или не превышало значение свойства ListCount, иначе могут возникнуть ошибки при установке нового значения.

Примечание. *Высоту элемента списка в твипах ItemHeightTwips& можно вычислить один раз вне цикла, например при загрузке формы, а потом передавать ее в процедуру List1_MouseMove в качестве общего параметра формы.*

```
Option Explicit
Private Declare Function SendMessage Lib _
    "user32" Alias "SendMessageA" (ByVal hWnd _
    As Long, ByVal wMsg As Long, ByVal wParam _
    As Long,  lParam As Any) As Long
  Const LB_GETITEMHEIGHT = &H1A1
```

[14] http://www.visual.2000.ru/develop/ms-vb/tips/9808.htm

```
Private Sub List1_MouseMove(Button As Integer, _
    Shift As Integer, X As Single, Y As Single)
  ' Выделение текущей позиции списка в
  ' соответствии с перемещением курсора мыши
  Dim ItemHeightPixels&, ItemHeightTwips&, _
  NewIndex&
  With List1
    ' Высота элемента в пикселах
    ItemHeightPixels& = SendMessage(.hWnd, _
    LB_GETITEMHEIGHT, 0, 0)
    ' Высота элемента в твипах
    ItemHeightTwips& = ScaleY(ItemHeightPixels&, _
    vbPixels,  vbTwips)
    ' Вычисление индекса
    NewIndex& = .TopIndex + (Y \ ItemHeightTwips&)
   ' Проверка — не вышли ли мы за пределы списка?
   If NewIndex& < .ListCount Then .ListIndex = NewIndex&
  End With
End Sub
```

Совет 157. Применяйте повторно используемые процедуры

Вернемся к предыдущему совету. А если такой режим работы со списком понадобится применить и для других списков? Писать такой код в каждой процедуре List_MouseMove?

Нет, нужно создать отдельный программный модуль, например ListMenu.Bas (или использовать уже имеющийся модуль, где вы храните свои повторно используемые вспомогательные процедуры), и сформировать в нем подпрограмму (ListMenuMouse) на основе программного кода подпрограммы List1_MouseMove. Текст такого модуля приведен в листинге 1. Обратите внимание на некоторые изменения в программе, в частности на то, что в качестве параметров в эту универсальную подпрограмму передается не только описание элемента управления, но и формы. Это необходимо, так как на самом деле метод ScaleY должен быть привязан к конкретной форме. В модуль ListMenu.Bas переместились также описания используемой API-функции и константы.

Листинг 1

```
Attribute VB_Name = "ListMenu"
Option Explicit

Private Declare Function SendMessage Lib "user32"_
   Alias "SendMessageA" (ByVal hWnd As Long, _
   ByVal wMsg As Long, ByVal wParam _
   As Long, lParam As Any) As Long
Const LB_GETITEMHEIGHT = &H1A1

Public Sub ListMenuMouse(cl As Control, fm As Form, _
   Y As Single)
   '
   ' Выделение текущей позиции списка
   ' в соответствии с перемещением курсора мыши
   Dim ItemHeight As Long, NewIndex As Long
   With cl
       ' Высота элемента списка в пикселах
       ItemHeight = SendMessage(.hWnd, LB_GETITEMHEIGHT, _
           0, 0)
       ' Преобразование из пикселов в твипы
       ItemHeight = fm.ScaleY(ItemHeight, vbPixels, vbTwips)
```

```
        ' Вычисление индекса элемента списка
        NewIndex = .TopIndex + (Y \ ItemHeight)
        ' Проверка — не вышел ли индекс за пределы списка?
        If NewIndex < .ListCount Then .ListIndex = NewIndex
    End With
End Sub
```

Теперь этот модуль можно подключить к какому-либо проекту, и, чтобы использовать режим отслеживания движения мыши в любом из его списков, нужно просто вставить следующую строку в событие ListN_MouseMove (заменив ListN на конкретное имя элемента управления):

```
Call ListMenuMouse(ListN, Me, Y)
```

Совет 158. Как очистить все текстовые окна

Чтобы очистить все текстовые окна на форме, используйте такой код:

```
Dim vControl
For Each vControl In Me.Controls
  ' где Me — текущая форма
  If TypeOf vControl Is TextBox Then
    vControl.Text = ""
  End If
Next
```

Совет 159. Используйте клавиатуру для управления размерами и расположением элементов управления

Иногда бывает гораздо удобнее использовать не мышь, а клавиатуру — стрелки Right, Left, Down и Up (Вправо, Влево, Вниз и Вверх) — для перемещения границ элементов управления. Для изменения размеров элементов управления (левый верхний угол остается на том же месте, перемещаются только правая и нижняя границы) надо использовать соответствующие клавиши стрелок при нажатой Shift. А для перемещения всего объекта, без изменения его размеров, следует нажать клавишу Ctrl.

Для того чтобы переместить левую (или верхнюю) границу элемента управления, например, вправо, нужно сначала сместить весь объект (Ctrl+Right), а потом вернуть на исходное место правую границу (Shift+Left).

Если вы выделите на форме сразу несколько элементов управления (для этого нужно сначала выделить первый из них с помощью мыши, а потом все остальные — щелкая их мышью при нажатой клавише Shift), то операции изменения размеров и их положения будут выполняться сразу для всех этих объектов.

Совет 160. Эмуляция события Click для правой кнопки мыши

Иногда бывает полезно использовать щелчок правой кнопкой мыши при работе с командной кнопкой. Например, для выполнения некоторых действий программы, отличных от обычного щелчка левой кнопкой. Для этого можно использовать события MouseDown и MouseUp, которые реагируют на действия с обеими кнопками мыши и позволяют определять, с какой из кнопок конкретно произошло событие. Однако здесь есть некоторые подводные камни, которые следует иметь в виду.

Обратите внимание на специфику реализации события Click для командной кнопки. Данное событие происходит в тот момент, когда вы отпускаете кнопку мыши, но при условии, если в моменты нажатия и отпускания левой кнопки мыши (а время между ними может быть достаточно большим) курсор мыши находится в пределах изображения командной кнопки. Понятно, что событие MouseDown сразу не подходит для эмуляции Click. Что же касается события MouseUp, то оно происходит, даже если в этот момент курсор мыши уже ушел за пределы кнопки — главное, чтобы он был в ее пределах в момент события MouseDown.

Решение задачи достаточно очевидно — нужно определить координаты мыши в событии MouseUp и проверить, попадают ли они в пределы изображения кнопки. Но при этом нужно иметь в виду, что координаты курсора мыши имеют значение относительно данного элемента управления, а не формы. Сделать это можно следующим образом:

```
Sub Command1_MouseUp (Button As Integer, _
  Shift As Integer, X As Integer, Y As Integer)
  '
  If Button = 2 Then  ' Правая кнопка мыши
    If X >= 0 And X <= Command1.Width And _
      Y >= 0 And Y <= Command1.Height Then
      ' Здесь нужно написать код, который
      ' обрабатывает данное событие
    End If
  End If
End Sub
```

Совет 161. Установка курсора мыши в нужное место на форме

Отличительной чертой коммерческих приложений является то, что курсор может "прыгать", то есть автоматически перемещаться на кнопку в диалоговом окне, которая предлагается по умолчанию. С помощью простой API-функции вы можете сделать то же самое и в своих приложениях:

```
  #If Win32 Then
    Private Declare Function SetCursorPos Lib _
      "user32" (ByVal x As Long, ByVal y As Long) As Long
  #Else
    Declare Sub SetCursorPos Lib "User" _
    (ByVal x As Integer, ByVal y As Integer)
  #End If
Private Sub Form_Load()
  #If Win32 Then
    Dim x As Long, y As Long
  #Else
    Dim x As Integer, y As Integer
  #End If
  '
  Me.Show
  ' Вычисляет координаты командной кнопки
  x = (Me.Left + Command1.Left + _
  Command1.Width / 2) / Screen.TwipsPerPixelX
  y = (Me.Top + Command1.Top + _
  Command1.Height / 2 + Me.Height - _
  Me.ScaleHeight) / Screen.TwipsPerPixelY
  ' Помещает курсор на кнопку
  SetCursorPos x, y
End Sub
```

Событие Form_Load вычисляет экранные координаты центра командной кнопки в пикселах. Затем функция SetCursorPos перемещает курсор мыши в требуемое место, аналогично функции "Snap-To", реализованной в продуктах Microsoft. Обратите внимание, что координаты и линейные размеры элементов управления задаются в твипах, а координаты курсора мыши при обращении к API-функции — в пикселах.

Совет 162. Используйте массивы элементов управления

Иногда бывает очень удобно объединить в массив группу элементов управления, выполняющих некоторые однотипные операции (мы будем это делать в последующих советах на примере командных кнопок). Тогда такой массив выступает в качестве одного комплексного объекта, а идентификация конкретного его элемента выполняется с помощью свойства Index (нумерация от нуля). За массивом закрепляется набор событийных процедур, состав которых аналогичен процедурам для отдельных элементов управления, но только в качестве первого параметра там будет передаваться номер индекса, по которому можно определить, с каким же конкретно элементом произошло событие.

Обращение к свойствам и методам отдельного элемента также выполнятся с помощью индекса, например cmdAction(Index).Caption = "Кнопка". Но у самого массива (без индекса) есть собственный набор свойств и методов.

Создается массив достаточно просто — в него включаются элементы с одинаковыми именами (свойство Name). При этом индекс формируется автоматически, но тут есть некоторые любопытные моменты. Например, вы последовательно сформировали четыре кнопки с именем cmdAction, у которых свойство Caption установили как А, Б, В и Г соответственно. Они сразу же получили индексы 0, 1, 2, 3.

Если теперь вы удалите кнопку Б, то индексы других кнопок не изменятся (не будет элемента с индексом 1). Теперь добавьте новую кнопку Д — она получит первый свободный индекс, равный 1. Далее мы хотим изменить индекс кнопки В. VB разрешит ввести только неиспользуемое значение индекса, например 4 или 10, то есть значение индекса конкретного элемента, присвоенного ему при создании, может быть изменено только самим программистов в явном виде — в отличие от свойства TabIndex (порядок обхода элементов управления на форме при нажатии клавиши Tab), где перенумерация выполняется автоматически.

Совет 163. Код, расположенный после вызова события Unload Me, препятствует закрытию формы

Вы можете столкнуться с такой проблемой при попытке выгрузки формы. Например, вы создали массив кнопок, с помощью которых хотите управлять представлением формы на экране, причем одна из них использует событие Unload Me для выгрузки формы. Для этого вы пишете следующий код:

```
Private Sub cmdAction_Click(Index As Integer)
'
' *** эта подпрограмма не работает!
'
  Select Case Index
    Case 0: Me.WindowState = vbMaximized
    Case 1: Me.WindowState = vbNormal
    Case 2: Me.WindowState = vbMinimized
```

```
    Case 3: Unload Me ' не работает данная строка!
  End Select
  MsgBox "Текущее состояние: " & _
  CStr(Me.WindowState)
End Sub
```

Однако приведенная выше подпрограмма не работает, поскольку за оператором Unload Me идет строка, содержащая исполняемый код. Таким образом, форма станет невидимой, но она не будет выгружена. Чтобы исправить эту ошибку, необходимо заменить вызов события Unload Me на установку переменной и выполнить это событие в самой последней строке подпрограммы:

```
Private Sub cmdAction_Click(Index As Integer)
  '
  ' *** эта подпрограмма работает!
  '
  Dim blnUnload As Boolean
  Select Case Index
    Case 0: Me.WindowState = vbMaximized
    Case 1: Me.WindowState = vbNormal
    Case 2: Me.WindowState = vbMinimized
    Case 3: blnUnload = True
  End Select
  MsgBox "Текущее состояние: " & _
  CStr(Me.WindowState)
  If blnUnload = True Then Unload Me
End Sub
```

Андрей Колесов, Ольга Павлова

© 1998, Андрей Колесов, Ольга Павлова
Авторский вариант. Вторая часть статьи, опубликованой с незначительной литературной правкой в журнале "КомпьютерПресс" N 01/99, компакт-диск.

Совет 164. Как прочитать имена стандартных каталогов

В составе Win32 API есть функции, с помощью которых можно узнать имена каталогов, необходимых часто для работы приложения: текущего (где находится само приложение), временного (для хранения временных файлов), операционной системы (Windows) и системного (System). (Не совсем понятно, зачем нужно иметь отдельную функцию для определения системного каталога - он всегда имеет имя \System внутри основного каталога ОС.)

Вариант подпрограммы StandardDirNameAPI$, которая выполняет эти операции, приведен в листинге 2 (модуль SDirName.BAS). По поводу этого программного кода можно сделать еще несколько замечаний. Во-первых, видно, что все четыре API-функции совершенно однотипны, но почему-то в двух из них параметры (имя строкового буфера и его длина) записаны в одном порядке, а в других - в обратном. Во-вторых, при обращении к GetTempPath имя каталога возвращается с символом "\" в конце, в остальных функциях - без него. Именно поэтому в процедуре StandardDirNameAPI$ было бы полезно использовать код, который приводил бы имя каталога в некоторый стандартный вид.

Листинг 2

```
Attribute VB_Name = "SDirName"

Option Explicit

Declare Function GetCurrentDirectory Lib "kernel32" _
   Alias "GetCurrentDirectoryA" (ByVal nBufferLength _
   As Long, ByVal lpBuffer As String) As Long
Declare Function GetTempPath Lib "kernel32" Alias _
   "GetTempPathA" (ByVal nBufferLength As Long, ByVal _
   lpBuffer As String) As Long
Declare Function GetSystemDirectory Lib "kernel32" _
   Alias "GetSystemDirectoryA" (ByVal lpBuffer As _
   String, ByVal nSize As Long) As Long
Declare Function GetWindowsDirectory Lib "kernel32" _
   Alias "GetWindowsDirectoryA" (ByVal lpBuffer As _
   String, ByVal nSize As Long) As Long

Public Static Function StandardDirNameAPI$(Index%)
   '
   ' Определение имен стандартных каталогов Windows
   '
   ' Длина буфера для имени каталога
   Const BufferLength& = 256
   Dim Result&, Buffer$, DirName$
   Buffer$ = Space$(BufferLength)
   Select Case Index%
       Case 0 ' Текущий каталог
           Result& = GetCurrentDirectory(BufferLength&, _
               Buffer$)
       Case 1 ' Временный каталог
           Result& = GetTempPath(BufferLength&, Buffer$)
       Case 2 ' Каталог Windows
           Result& = GetWindowsDirectory(Buffer$, _
               BufferLength&)
       Case 3 ' Системный каталог
           Result& = GetSystemDirectory(Buffer$, _
               BufferLength&)
   End Select
   '
   DirName$ = Left$(Buffer$, Result&)
   ' Убрать "\" справа в имени каталога
   'If Right$(DirName$, 1) = "\" Then DirName$ = _
       ' Left$(DirName$, Result& - 1)
    или наоборот добавить "\"
   If Right$(DirName$, 1) <> "\" Then DirName$ = _
       DirName$ + "\"
   StandardDirNameAPI$ = DirName$
End Function
```

В результате мы формулируем еще один общий СОВЕТ: "Разработчики (в том числе и создатели Win API)! Общайтесь между собой и на рабочие темы!"

Совет 165. Определитесь - что такое текущий и временный каталоги?

Это совет вытекает из предыдущего и является его продолжением. Для тестирования процедуры StandardDirNameAPI$ мы предлагаем создать небольшое приложение Tip164.vbp в виде формы, на которой расположен массив из четырех командных кнопок, двух меток и одного текстового поля (рис. 3) и подключенного модуля SDirName.BAS.

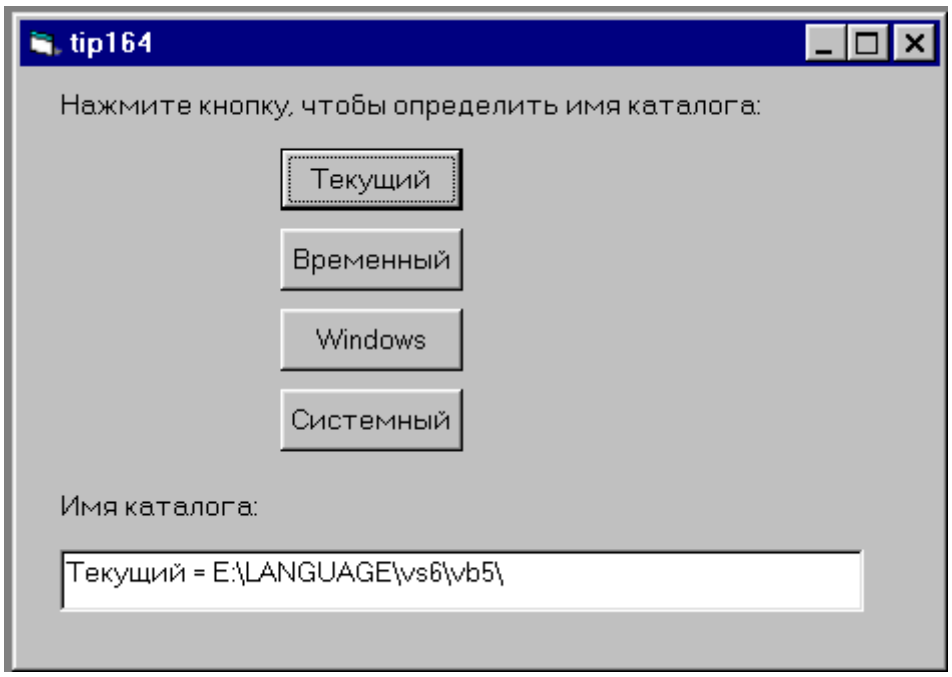


Рис. 3

Весь программный код формы состоит из одной событийной процедуры:

```
Private Sub cmdAction_Click(Index As Integer)
  ' Определение имени каталога
  Dim DirName$
  DirName$ = StandardDirNameAPI$(Index)
  If DirName$ = "" Then _
    DirName$ = " Не смогли определить "
    Text1.Text = cmdAction(Index).Caption + " = " + _
                 DirName$
End Sub
```

Если нумерация кнопок (сверху вниз) правильно согласуется с их названиями, то после щелчка командной кнопки в текстовом поле будет появляться имя соответствующего каталога. Теперь начнем наше исследование.

Что такое каталоги Windows и System - в целом понятно. Их имена раз и навсегда жестко фиксируются в момент инициализации Windows. С текущим каталогом дела обстоят посложнее. Это каталог, где содержится запущенное на выполнение приложение. Соответственно, если вы запускаете Tip164.vbp в среде VB, то текущим будет каталог, где находится VB. Если же вы создадите и запустите автономное приложение Tip164.exe, то текущим будет каталог, где размещается этот файл.

А что такое временный каталог? Вопрос довольно актуальный, лично нас всегда поражало какое-то не поддающееся простой логике хаотическое появление временных файлов в различных разделах диска.

В литературе по Windows 95 встречается такое определение: "Это каталог, описанный переменной окружения TMP. Если переменная TMP не задана - то переменной TEMP. Если TEMP также отсутствует, то тогда это текущий каталог приложения". Однако наши эксперименты показали, что эта формулировка не верна.

Действительно, временным является каталог, описанный переменной окружения TMP (например, в AUTOEXEC.BAT). Но если TMP не задана, то временным является подкаталог \TEMP в каталоге операционной системы (Windows). Причем, если его не было раннее, то он создается автоматически. Но это еще не все.

Предположим, у вас есть несколько логических дисков и вы в качестве временного почему-то решили определить корневой каталог диска D:. Если вы запишете SET TMP=D:\, то все будет OK. Но если забыли в конце строки "\", то тут нужно разбираться отдельно.

Так вот, в последнем случае определение временного каталога зависит от местонахождения проекта Tip164. Если он расположен вне диска D:, то временным каталогом будет D:\. Если же он находится на самом диске D:, то временным будет... каталог самого проекта (этот случай приведен на рис. 3). Причем независимо от того, запущен Tip164 из среды VB или как автономный файл.

Короче говоря, будет время - поэкспериментируйте. Вы обнаружите много любопытных вещей и еще раз подивитесь витиеватости замыслов разработчиков Microsoft Windows.

Совет 166. Включение/выключение всех элементов управления в массиве

Как известно, VB не позволяет передавать массив элементов управления в качестве аргумента в подпрограмму или функцию. И тем не менее вполне возможно придумать решение этой проблемы. Скажем, вы хотите включать/отключать любой набор элементов управления на форме. Это можно сделать с помощью следующей подпрограммы, помещенной в отдельный в модуль:

```
Public Sub EnableControls(Form As Form, _
  ControlArray As Control, State As Boolean)
  '
  Dim ctl As Control
  For Each ctl In Form.Controls
    If ctl.Name = ControlArray.Name Then
      ' Данный элемент управления является
      ' элементом массива
      ctl.Enabled = State
    End If
  Next
End Sub
```

Чтобы проверить ее работу, создайте форму и разместите на ней массив полей текста с именем txtSample и массив флажков chkSample. Добавьте две командные кнопки (cmdText и cmdCheck) и для каждой из них в событие Click введите соответственно два таких кода:

```
Call EnableControls(Me, txtSample(0), Not (txtSample(0).Enabled))

Call EnableControls(Me, chkSample(0), Not (chkSample(0).Enabled))
```

Запустите на выполнение ваше приложение и по очереди щелкните кнопки, чтобы включить/отключить присоединенные группы элементов управления.

Совет 167. Сортировка пронумерованных элементов в окне списка

Предположим, что у вас есть пронумерованные элементы (в нашем случае - файлы):

```
FILE1.BMP FILE2.BMP FILE3.BMP FILE10.BMP
```

Если вы поместите их в элемент управления ListBox или ComboBox, то после сортировки они будут представлены в списке следующим образом:

```
FILE1.BMP
FILE10.BMP
FILE2.BMP
FILE3.BMP
```

А вы хотите, чтобы они выводились так:

```
FILE1.BMP
FILE2.BMP
FILE3.BMP
FILE10.BMP
```

Для этого можно использовать подпрограмму ReSort (листинг 3). После того как вы заполните окно списка, вызовите ReSort, передав исходный элемент управления ListBox или ComboBox в качестве единственного параметра, например Call ReSort(List1).

Листинг 3

```
Sub ReSort(L As Control)
    '
    Dim P%, PP%, C%, Pre$, S$, V&, NewPos%, CheckIt%
    Dim TempL$, TempItemData&, S1$
    '
    For P = 0 To L.ListCount - 1
        S = L.List(P)
        For C = 1 To Len(S)
            V = Val(Mid$(S, C))
            If V > 0 Then Exit For
        Next
        If V > 0 Then
            If C > 1 Then Pre = Left$(S, C - 1)
            NewPos = -1
            For PP = P + 1 To L.ListCount - 1
                CheckIt = False
                S1 = L.List(PP)
                If Pre <> "" Then
                    If InStr(S1, Pre) = 1 Then _
                        CheckIt = True
                Else
                    If Val(S1) > 0 Then CheckIt = True
                End If
                If CheckIt Then
                    If Val(Mid$(S1, C)) < V Then _
                        NewPos = PP
                Else
                    Exit For
                End If
            Next
            If NewPos > -1 Then
                TempL = L.List(P)
                TempItemData = L.ItemData(P)
                L.RemoveItem (P)
                L.AddItem TempL, NewPos
                L.ItemData(L.NewIndex) = TempItemData
                P = P - 1
```

```
            End If
        End If
    Next
End Sub
```

Совет 168. Убедитесь, что вы закрыли все объекты доступа к данным при выходе из программы

Если вы используете какие-либо объекты доступа к данным (DAO, RDO или ADO) в своей программе, следует перед выходом из нее закрыть все открытые наборы записей, базы данных и рабочие области. Конечно, когда вы выходите из программы, все указатели к этим объектам автоматически уничтожаются. Но если же вы не закроете в явном виде все открытые элементы, то связи с базой данных освободятся не сразу, а память, используемая этими объектами, может так никогда и не быть перераспределена операционной системой.

Ниже дается короткая подпрограмма, которую вы можете добавить к своему событию Form_Unload (или любому другому модулю, завершающему выполнение программы) и которая закрывает все открытие рабочие области, базы данных и наборы записей для объекта доступа к данным (DAO) и освобождает память, резервируемую этими объектами. Приведенный здесь код будет работать всегда при выходе из формы независимо от того, сколько связей у вас будет открыто - одна, сто или даже ни одной.

```
Private Sub Form_Unload(Cancel As Integer)
  '
  ' Закрывает все объекты доступа к данным и
  ' освобождает всю память
  '
  On Error Resume Next
  Dim ws As Workspace
  Dim db As Database
  Dim rs As Recordset
  '
  For Each ws In Workspaces
    For Each db In ws.Databases
      For Each rs In db.Recordsets rs.Close
         Set rs = Nothing
      Next
      db.Close
      Set db = Nothing
    Next
    ws.Close
    Set ws = Nothing
  Next
End Sub
```

Совет 169. Проверка наличия файла из любого места программы

В приложениях довольно часто приходится делать проверку наличия определенных файлов. Здесь приводится простая подпрограмма, которая предназначена для выполнения этого:

```
Public Function VerifyFile(FileName$) As Boolean
  ' Проверка - существует ли указанный файл
  On Error Resume Next
  ' Файл открывается как входной, последовательный
  Open FileName$ For Input As #1
```

```
  If Err Then ' Ошибка при открытии - нет файла
    VerifyFile = False
  Else
    VerifyFile = True: Close #1
  End If
End Function
```

Для проверки ее работоспособности разместите на форме текстовое поле и командную кнопку, для которой напишите такой код в событии Click:

```
Private Sub Command1_Click()
  Dim FileName$, a$
  FileName$ = Text1.Text
  If VerifyFile(FileName$) Then _
    a$ = "" Else a$ = " не"
  MsgBox ("Файл " & FileName$ & a$ & " найден")
End Sub
```

Щелкнув кнопку, вы получите сообщение о наличии файла, имя которого было набрано в текстовом поле.

Совет 170. Простой способ генерации случайных чисел

Если вашей программе часто требуются случайные числа, вам, вероятно, надоело каждый раз вставлять команду Randomize, инициализирующую датчик случайных чисел, а после нее - уравнение. Приведенная здесь простая подпрограмма будет выполнять обе эти операции. Вначале поместите следующую функцию в один из модулей своего проекта:

```
Public Function GenRndNumber(Lower%, Upper%)
  Randomize
  GenRndNumber = Int((Upper% - Lower% + 1) * Rnd + Lower%)
End Function
```

Чтобы получить случайное число в диапазоне от -99 до 99, просто введите:

```
RandomNumber = GenRndNumber(-99, 99)
```

Вы можете также получить случайную букву в диапазоне от A до M следующим образом:

```
RandomLetter = Chr$(GenRndNumber(Asc("A"), Asc("M")))
```

Кроме того, можно удалить середину диапазона, поместив вызов к функции GenRndNumber внутри цикла Do...Loop. Так вы получите случайное число в диапазоне от 50 до 99 или от -50 до -99:

```
' Инициализирует число, которое будет сгенерировано
' внутри области, которую вы хотите исключить
RandomNumber = 0
Do Until Abs(RandomNumber) > 49
  RandomNumber = GenRndNumber (-99, 99)
Loop
```

Однако использовать такую универсальную процедуру следует только тогда, когда время ее выполнения не очень критично для программы. В случае же, когда вычисление случайных чисел нужно выполнять в очень больших объемах, этот код лучше вставить внутрь соответствующей программы.

Андрей Колесов, Ольга Павлова

© 1999, Андрей Колесов, Ольга Павлова
Авторский вариант. Статья была опубликована с незначительной литературной правкой в журнале "КомпьютерПресс" N 02/99, компакт-диск

Совет 171. Используйте функцию CopyMemory из Win32 API

При разработке приложений довольно часто встречается простая задача пересылки N последовательных байтов из одной области оперативной памяти в другую. В VB проблема возникает, когда такая область памяти определена не именем переменной, а в виде адреса (указателя — Pointer). К сожалению, встроенные средства VB не позволяют работать с адресами, а порой это крайне необходимо.

Так, некоторые функции Win API, используемые для получения информации о ресурсах системы (например, VerQueryValue), возвращают в качестве параметра адрес области памяти FiAddr&, где лежит искомая информация. Но как переписать эти данные, к примеру, в целочисленный массив для дальнейшей обработки в среде VB? Это можно сделать с помощью функций Win API.

В составе Win16 API для этого имелась функция HMemCpy:

```
Declare Sub HMemCpy Lib "kernel"_
  (hpvDest As Any, hpvSource As Any, ByVal cbCopy As Long)
```

Она появилась только в расширенном варианте функций в Windows 3.1 (в версии 3.0 ее не было), а в пакете VB 3.0 ее описание было приведено только в файле WIN31WH.HLP.

Что же касается Win32 API, то в описании WIN32API.TXT для VB 4.0 и 5.0 о функции HMemCpy или ее аналоге даже и не упоминается. Более того, в книге Дэна Эпплмана по Win32 API, на которую мы часто ссылаемся (о ней мы писали в Совете 133), ничего не говорится о подобной функции. Однако, поскольку такая функция действительно очень нужна, Дэн предлагает использовать процедуру agCopyData из его собственной библиотеки APIGI32.DLL, которая приводится на прилагаемом компакт-диске.

Все это выглядит весьма странно. Более тщательный поиск, проведенный с помощью специалиста московского отделения Microsoft Юрия Томашко, увенчался успехом — в Win32 есть функция копирования байтов CopyMemory и ее описание появилось в VB 6.0 в файле WIN32API.TXT. Почему Microsoft скрывает от пользователей VB 4.0 и 5.0 ее существование, и почему Дэн Эпплман хранит эту тайну — можно только гадать. Кстати, настоящее имя функции в библиотеке KERNEL32 — RtlMoveMemory:

```
Public Declare Sub CopyMemory Lib _
  "kernel32" Alias "RtlMoveMemory" _
  (Destination As Any, Source As Any, _
  ByVal Length As Long)
```

Вот типичный пример применения этой функции:

```
Di& = VerQueryValue(VerBuf(0), "\", FiAddr&, FiLen&)
' Возвращаются:
```

```
' FiAddr& - адрес области с информацией
' о версии файла
' FiLen& - длина области в байтах
CopyMemory ByVal FiAddr&, Ffi, 52
' Ffi - структура данных для информации о версии
' 52 - ее длина
```

CopyMemory можно использовать для создания очень полезных функций преобразования типов данных (см. Совет 172), а также для ускорения в сотни и даже тысячи раз выполнения операций, реализуемых традиционными средствами VB. Например, для копирования целых массивов или их фрагментов:

```
Dim Arr1!(1000), Arr2!(2000), i%
For i = 1 To 1000: Arr1(i) = i + 0.1: Next
' Вместо:
' For i = 1 To 1000: Arr2(1000 + i) = Arr1(i): Next
' Использовать:
CopyMemory Arr2(1001), Arr1(1), 1000 * 4
'
MsgBox ("result =" & Arr2(1600))
```

Совет 172. Копирование областей памяти в DOS

Дополнительные функции DLL-библиотек могут серьезно расширить возможности VB-программиста. При этом следует иметь в виду, что для написания таких процедур зачастую совсем не обязательно быть большим знатоком языка, на котором они будут писаться.

Например, функция копирования областей памяти пригодится и тем, кто еще работает в Basic/DOS. В силу специфики использования библиотек в этих версиях Basic (мы вновь сожалеем, что в VB/Win-проектах Microsoft не позволяет подключать к исполняемому модулю объектные библиотеки) такие внешние функции лучше всего было писать на Ассемблере. Посмотрите, какой простой код имеет функция StringCopy, написанная для варианта MASM 6.0 и фактически являющаяся точным аналогом функции CopyMemory для режима DOS (только число байтов задается целочисленной переменной):

```
.MODEL Medium,Basic
.CODE
StringCopy PROC USES DS DI SI DF,
SourceAddr:DWord, DestAddr:DWord, Len:Word
  ; прием входных параметров:
  MOV CX,Len  ; количество байт
  LES DI,DestAddr ; полный адрес Приемника (Куда)
  LDS SI,SourAddr ; полный адрес Источника (Откуда)
  ; пересылка данных:
  CLD ; очистка флага DF
  REP MOVSB  ; пересылка CX-байт
  ; выход из процедуры:
  RET ; возврат управления
  StringCopyByv ENDP
END
```

Ее описание можно сделать двумя способами:

1. Адреса задаются с помощью двух 16-разрядных переменных — сегмент и смещение:

   описание:

```
DECLARE SUB StringCopy(BYVAL SourceSeg%, BYVAL SourceOff%,_
BYVAL DistSeg%, BYVAL DistOff%, BYVAL LenByte%)
```

обращение:

```
CALL StringCopyByv(SourceSeg%, SourceOff%, _
DistSeg%, DistOff%, LenByte%)
```

2. Полные адреса задаются с помощью 32-разрядных переменных: описание:

```
3.  DECLARE SUB StringCopy(BYVAL SourceAdr&, BYVAL DistAdr&, _
4.  BYVAL LenByte%)
```

обращение:

```
CALL StringCopyByv(SourceAddr&, DistAddr&, LenByte%)
```

Совет 173. Как реализовать функции MKx$/CVx в VB/Win

Именно с этого совета (N 3[15]) три года назад мы начали свои публикации для пользователей VB в КомпьютерПресс N 3'96. Напомнить о нем мы решили для иллюстрации применения функции CopyMemory.

Дело в том, что в DOS-овских версиях MS Basic (Quick, PDS, Visual) имелась группа очень полезных встроенных функций MKx$/CVx (x — тип числовых данных: I, L, S, D), которые почему-то пропали в VB/Win. (Их описание приводится во встроенной справке QBasic, которая входит в состав MS DOS 5.0 и 6.x.)

С помощью этих функций производится преобразование числовых (целых, вещественных и пр.) данных в строковый формат и наоборот. Точнее говоря, никакого преобразования значений здесь не выполняется, а просто N-е количество байт меняет название типа данных.

Основной смысл такого преобразования — это возможность хранения и передачи разнотипных данных в виде одной строковой переменной, что является отличной альтернативой структурам данных Type с их жестким описанием полей на уровне исходного текста. В свое время мы очень широко использовали в своей практике этот прием для создания гибких, динамически настраиваемых структур данных.

Применение данных функций позволяет осуществлять весьма изящные преобразования данных. Например, в Совете 117[16] мы говорили о проблеме обработки беззнаковых целых чисел и приводили пример слияния двух 16-разрядных целых чисел в 32-разрядное представление и наоборот.

Сравните приведенный там пример со следующим вариантом:

1. LongValue& => IntHigh% & IntLow%

```
2.  IntHigh% = CVI(LEFT$(MKL$(LongValue&), 2))
3.  IntLow% = CVI(RIGHT$(MKL$(LongValue&), 2))
4.  ' вместо Совета 117
5.  IntegetHigh% = LongValue& \ &H10000
6.  IntegerLow% = LongValue& And &H7FFF
```

[15] http://www.visual.2000.ru/develop/ms-vb/tips/9603.htm
[16] http://www.visual.2000.ru/develop/ms-vb/tips/9710.htm

```
7.   If (LongValue& AND &H8000) <> 0 Then
8.     IntegerLow% = IntegerLow% Or &H8000
9.   End If
10. IntHigh% & IntLow% => LongValue&
11.  LongValue& = CVL(MKI$(IntHigh%) + MKI$(IntLow%))
12.  ' вместо Совета 117
13.  LongValue& = IntHigh% * &H10000 + (IntegerLow% AND &H7FFF)
14.  If IntegerLow% < 0 Then LongValue& = LongValue& or &h8000&
```

С помощью функций Win API в VB/Win можно довольно просто реализовать эти полезные Basic-функции. Набор таких процедур приведен в модуле MKXCVX.BAS, а пример их применения — в модуле MKXCVXTS.BAS (см. листинг Lst173[17]).

Совет 174. Как сделать фон формы в виде палитры цветов

Хотите сделать фон своей формы в виде палитры цветов, вроде той, что Microsoft любит выдавать на экране при работе своих установочных утилит SETUP.EXE? Такой стиль раскрашивания называется градиентным заполнением и легко реализуется с помощью следующей процедуры:

```
Sub Dither(vForm As Form)
  Dim intLoop As Integer
  vForm.DrawStyle = vbInsideSolid
  vForm.DrawMode = vbCopyPen
  vForm.ScaleMode = vbPixels
  vForm.DrawWidth = 2
  vForm.ScaleHeight = 256
  For intLoop = 0 To 255
    vForm.Line (0, intLoop)-(Screen.Width, _
       intLoop - 1), RGB(0, 0, 255 -intLoop), B
  Next intLoop
End Sub
```

Теперь в событие Form_Activate соответствующей формы вставьте строку:

```
Dither ME
```

Цвет рисовки каждой полоски определяется с помощью функции RGB (Red-Green-Blue), основанной на смешении красного, зеленого и синего цветов. В приведенном выше примере закраска получается от черного до голубого цвета (как у Microsoft). Для черно-красного фона используйте такой вариант:

```
RGB(255 - intLoop, 0, 0).
```

А поклонникам красно-желтых цветов фирмы "1С" (рис. 1) следует применить:

```
RGB(255, 255 - intLoop, 0).
```

[17] http://www.visual.2000.ru/develop/ms-vb/tips/lst173.htm

Рис. 1

Совет 175. Помните о свойстве KeyPreview для формы

Как многие другие элементы управления, форма имеет стандартные события для обработки нажатия клавиш — KeyDown, KeyPress и KeyUp (об особенностях использования этих процедур и кодов клавиатуры см. Совет 129[18]. Их можно применять для управления кодами клавиш на уровне формы для всех находящихся на ней элементов управления.

Для этого следует сначала установить свойство KeyPreview формы как True. Тогда можно будет выполнять перехват всех событий KeyXX на уровне формы: в первую очередь будут выполняться процедуры формы, а уже потом — элементов управления.

Например, одна процедура Form_KeyPress может управлять режимом ввода во всех текстовых полях данной формы. А процедура Form_KeyUp может отслеживать нажатие "горячих" клавиш вне зависимости от того, где находится курсор (лучше использовать ее, а не KeyDown, так как традиционно считается, что команда выполняется при отжатии клавиши).

Если вам необходимо заблокировать обработку операций с клавишами в элементах управления (а они выполняются вслед за обработкой событий формы), установите значения KeyAscii=0 и KeyCode=0 в событиях KeyPress и KeyDown/Up соответственно.

Однако следует иметь в виду, что некоторые элементы управления (когда они находятся в фокусе) перехватывают определенные операции с клавишами в любом случае, и эти события не доходят до процедур формы. Например, командная кнопка всегда реагирует на нажатие Enter, а списки — на клавиши управления курсором.

Совет 176. Использование экзотических "быстрых" клавиш в меню

Иногда возникает необходимость присвоить элементу меню "быструю" клавишу, отличающуюся от той, что предлагается редактором меню. Например, для команды Exit вы можете захотеть использовать такую сложную комбинацию клавиш — Ctrl+Shift+Alt+Q. Для этого введите следующий код в событие Form_Load для формы:

[18] http://www.visual.2000.ru/develop/ms-vb/tips/9807.htm

```
Private Sub Form_Load()
  mnuExit.Caption = mnuExit.Caption & vbTab _
  & "Ctrl+Shift+Alt+Q"
End Sub
```

Данный код добавляет текст "Ctrl+Shift+Alt+Q" к названию элемента меню mnuExit и выравнивает его по правому краю относительно других "быстрых" клавиш в меню. Далее вспомните о предыдущем совете и установите свойство KeyPreview для формы как True, а затем напишите код для события KeyUp:

```
Sub Form_KeyUp (KeyCode As Interger, Shift As Integer)
  If KeyCode = 81 And Shift = 7 Then
    ' операция по команде Exit
  End If
End Sub
```

Совет 177. Используйте свойства Default и Cancel для командных кнопок

Довольно часто на форме располагаются две кнопки, обычно связанные с ее закрытием: Ok — выполнение некоторых действий, заданных формой, и Cancel — отмена каких-либо действий. При работе с клавиатурой подобные операции традиционно выполняются с помощью клавиш Enter и Esc. Чтобы задействовать использование этих клавиш, можно использовать свойства Default и Cancel для командных кнопок.

Если вы установите свойство Default для кнопки как True, то в любой момент работы с формой при нажатии Enter будет выполняться событие Click данной кнопки. Аналогично, если установить свойство Cancel как True, при нажатии Esc будет выполняться событие Click. Во избежание путаницы VB автоматически следит за выполнением двух правил:

1. Свойство Default или Cancel может быть установлено как True только для одной кнопки на форме.

2. Одна кнопка может иметь одновременно только одно свойство, установленное как True, — либо Default, либо Cancel.

Совет 178. Как определить имя накопителя CD-ROM

Если вам нужно определить имя накопителя на компакт-дисках, можете воспользоваться таким программным кодом:

```
Declare Function GetDriveType Lib "kernel32" _
  Alias "GetDriveTypeA" (strDrive As String) As Long
  Const DRIVE_CDROM = 5
Public Function GetCDROMDrive() As String
  Dim lType As Long
  Dim i As Integer
  Dim tmpDrive As String
  Dim found As Boolean
  '
  ' Просмотр по всем буквам A-Z:
  For i = 0 To 25
    tmpDrive = Chr(65 + i) & ":\"
    ' обращение к функции Win32 API:
    lType = GetDriveType(tmpDrive)
    If (lType = DRIVE_CDROM) Then
      ' найдет CD-ROM
```

```
      found = True: Exit For
    End If
  Next
  If Not found Then tmpDrive = ""
  GetCDROMDrive = tmpDrive
End Function
```

Совет 179. Как избежать ненужного обновления наборов записей

Приведенный здесь код пригодится для уменьшения влияния операции по обновлению наборов данных на ввод данных с клавиатуры. Для этого поместите на форму таймер (tmr_Timer) и установите свойство Interval как 1000 и свойство Enabled как False. Затем введите следующий код в событие txtFilter_Change текстового окна:

```
Private Sub txtFilter_Change()
  Timer1.Enabled = False
  Timer1.Enabled = True
End Sub
```

В событии Timer вызовите подпрограмму, которая обновляет ваш набор записей:

```
Private Sub Timer1_Timer()
  Timer1.Enabled = False
  Call MyUpdateRecordsetRoutine
End Sub
```

Теперь набор записей будет обновляться только в том случае, если вы не нажмете какую-либо клавишу в течение целой секунды. Каждый раз при нажатии клавиши будет происходить сброс таймера, а отсчет времени снова начинаться с нуля.

Совет 180. Упрощайте программный код

Конструкции, подобные следующей

```
If MyNumber > 32 Then
  BooleanValue = True
Else
  BooleanValue = False
End If
```

встречаются в программах довольно часто. Но гораздо привлекательнее выглядит такой вариант:

```
BooleanValue = (MyNumber > 32)
```

Совет 181. Будьте внимательны при работе с повторно используемыми BAS-компонентами в VB и VBA.

Этот совет появился в ходе подготовки материалов для нашей постоянной рубрики "Разработка приложений в среде MS Office 97" (см. VBA-Клуб[19]. Суть вопроса заключается в том, что с точки зрения программирования работа в Visual Basic и в Office/VBA выглядит почти тождественной. Однако на самом деле между ними есть

[19] http://www.microsoft.ru/offext/officedev/articles/kolesov/

немало принципиальных отличий, которые, в частности, касаются повторного использования ранее созданных процедур, написанных на VB.

Например, в статье "Разработка приложений с помощью Excel 97 и VBA. Часть 2" (КомпьютерПресс, ь 12'98) мы использовали несколько процедур из своих более ранних примеров. И именно на это мы хотим обратить сейчас ваше внимание. Отметим сразу, что данная тема непосредственно связана с детальным изучением компонентной структуры VB- и VBA-приложений.

Например, по ходу разработки приложения вам нужно использовать процедуру Proc1, которая хранится в модуле Module1.bas. Внешне эта задача решается в VB (автономном средстве разработки) и в VBA (точнее, в среде VBA некоторого конкретного офисного приложения, например Word) примерно одинаково: нужно загрузить этот модуль командой Project|Add Module или File|Import File соответственно. Но в этих операциях имеются и принципиальные отличия.

Как это происходит в Visual Basic

При работе с VB каждый используемый в нем BAS- и FRM-модуль продолжает оставаться автономным компонентом приложения, каждый хранится в виде отдельных файлов на диске. Собственно VB-проект — это совокупность автономных файлов с программным кодом, которые объединяются в один загрузочный EXE-модуль только в момент его создания. Разумеется, сейчас речь идет только о файлах BAS и FRM, которые могут формироваться непосредственно в среде VB и код которых помещается в загрузочный модуль. Другие компоненты приложения — OCX, DLL и пр. являются внешними и формируются отдельно.

Суть вопроса заключается в том, что создавая или корректируя BAS-модули (и FRM-файлы), вы автоматически изменяете файлы, хранящиеся на диске (обычно это делается в момент завершения работы пакета). Таким образом, если некоторые модули применяются сразу в нескольких проектах (то есть фактически являются повторно используемыми компонентами), сделанные изменения автоматически вносятся во все остальные приложения (разумеется, только в момент перекомпиляции исполняемого модуля).

Такая логика работы с исходными модулями кода (традиционная для всех систем программирования) имеет свои плюсы и минусы. Но общим выводом является необходимость уделять особое внимание именно повторно используемым компонентам, которые зачастую являются небольшими, но очень полезными вспомогательными процедурами.

Хорошим решением при работе в среде систем MS Basic для DOS было использование вспомогательных BAS-процедур в виде объектных (сейчас их часто называют статическими) LIB-библиотек, которые потом включались в состав исполняемого модуля. Остается только сожалеть, что Microsoft почему-то упорно не хочет реализовывать такой удобный и простой вариант в своем Visual Basic для Windows. Однако, начиная с версии VB4, можно оформлять подобные процедуры в виде ActiveX-серверов (DLL или EXE), к которым могут обращаться любые программы, поддерживающие технологии ActiveX (в том числе и приложения MS Office 97).

Однако речь идет о двух разных технологиях: одно из важных различий между объектными и динамическими библиотеками заключается в том, что в первом случае в состав исполняемого модуля записываются только те процедуры, на которые имеются

ссылки. В варианте же DLL в исполняемый модуль вообще ничего не записывается, но при обращении к любой функции из такой библиотеки в память грузится весь файл целиком.

В то же время нужно иметь в виду то, что при использовании LIB-библиотеки производится загрузка не только кода процедуры, на которую имеется ссылка, но и всего объектного модуля, включая остальные записанные в нем процедуры. То же самое происходит и при загрузке повторно используемых компонентов в виде исходных модулей.

Мы об этом говорим потому, что хотим обратить внимание на то, что распределение процедур по модулям — особое искусство, и о нем можно говорить отдельно. Тут есть два крайних варианта: либо поместить все вспомогательные процедуры в один модуль, либо каждую процедуру записать в отдельный модуль. Первый вариант представляется изначально неверным (слишком большая избыточность кода для конкретного приложения), второй (который был в свое время классическим стилем программирования) тоже не оптимален, так как заставляет программиста работать с огромным числом файлов. Оптимальный вариант лежит где-то посередине, не говоря уже о том, что объединение взаимосвязанных процедур (например, на уровне общих данных) в один модуль бывает просто необходимым.

Кроме использования готовых процедур в виде загрузки модуля целиком, можно просто копировать код самой процедуры и переносить его из одного модуля в другой. Но этот способ представляется одним из самых неудачных (обычно его применяют начинающие программисты), так как это приводит к путанице в версиях процедур и сложностям в их поддержке (например, если обнаружили ошибку, которую надо исправить во всех задействованных проектах). Столь же плох и вариант с копированием всего модуля (или работа с его копией под другим именем.)

Как обстоит дело в Office 97

Тут дело обстоит совсем не так, как в VB.

BAS-модуль, загруженный в Office/VBA (это относится и к Word, и к Excel) командой File|Import File, автоматически теряет логическую связь с соответствующим исходным BAS-файлом, хранимым на диске, и становится сугубо внутренним компонентом данного приложения. Соответственно, все изменения, сделанные впоследствии в BAS-файле, никак не влияют на состояние уже загруженного модуля. И наоборот, коррекция кода, выполненная внутри VBA, никак не влияет на состояние исходного модуля. Если Вы хотите использовать созданный (или измененный) код такого модуля, его нужно специально записать на диск командной File|Export File. Все это относится и к FRM-модулям. Эти отличия при работе в VB и VBA видны уже из названий команд ввода-вывода модулей: Load/Save и Import/Export. А также в представлении модулей в окне проектов. В случае VB (рис. 2) BAS-модуль представлен и именем файла (в скобках), и его свойством Name (Attribute VB_Name в первой строке файла).

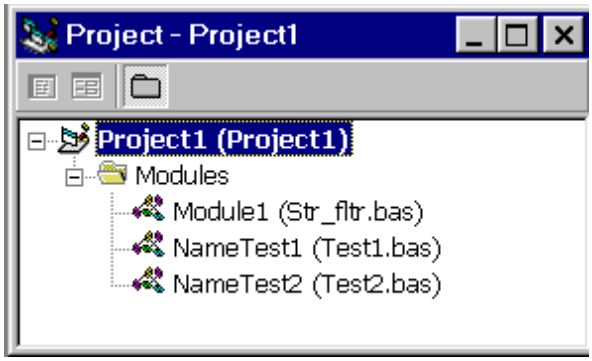


Рис. 2

Причем, если загружается BAS-модуль старой структуры (без Attribute VB_Name), то формируется стандартное имя ModuleX. При работе в VBA модуль идентифицируется только свойством Name без какого-то упоминания об исходном файле (рис. 3).

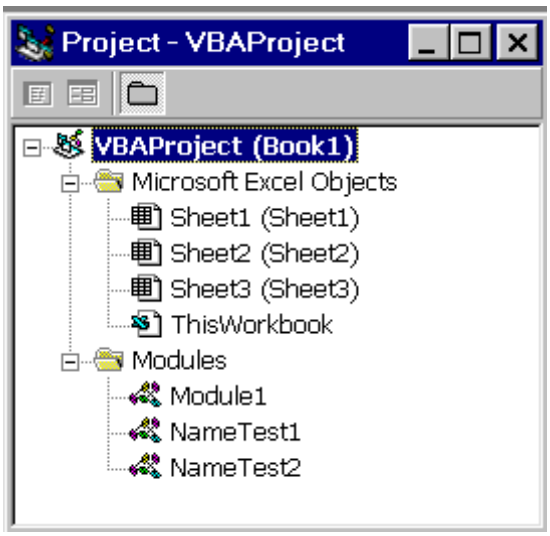


Рис. 3

Такой "внутренний" характер компонентов VBA-приложения в значительной степени изменяет технологию работы с повторно используемыми программными кодами. У программиста здесь фактически полностью развязаны руки в операциях перемещения процедур между модулями, удаления ненужных процедур и пр. При этом если работа с конкретным проектом упрощается, то технология поддержки повторно используемого инструментария существенно усложняется.

Выводы и рекомендации

1. Квалификация программиста во многом определяется его умением использовать готовые компоненты, а также формировать и пополнять набор таких средств из числа собственных разработок. У опытного программиста до 90% программного кода конкретного приложения состоит из готовых процедур, написанных им самим в предыдущие годы.

2. При разработке VB/VBA приложений четко разделяйте (хотя бы мысленно) повторно используемые модули и модули, которые вы собственно и пишете для работы данной программы.

3. Храните повторно используемые файлы в отдельных каталогах. Каждое приложение, а также его собственные (уникальные) исходные модули храните также в отдельных каталогах. Избегайте дублирования имен файлов, даже хранимых в разных каталогах. Тем более избегайте дублирования имен процедур. На вашем диске должна быть только одна рабочая копия программного кода.

4. Если вам захотелось использовать программный код, уже реализованный для другого приложения, то, видимо, имеет смысл оформить его в виде процедуры, которую можно включить в состав своих повторно используемых средств.

5. Избегайте создания процедур общего назначения в FRM-модулях (особенно в VB), кроме тривиальных случаев однозначной привязки программного кода к данной форме. Но лучше создайте BAS-модуль с тем же названием для хранения подобных процедур.

6. Тестирование и оформление (например написание комментариев) повторно используемых процедур нужно проводить особенно тщательно. Помните, что внесение в них изменений и отладка в рамках конкретного приложения еще не гарантируют их работоспособность в других программах (ведь ошибка может проявиться при специфическом наборе входных параметров).

7. Избегайте без особой нужды модернизации готовых процедур. Тем более не меняйте их входные спецификации. При работе с библиотеками LIB и DLL это может привести к аварийным ситуациям.

8. Внимательно относитесь к распределению процедур между отдельными BAS-модулями. Старайтесь не злоупотреблять перемещением процедур между различными файлами.

9. Для надежного сохранения сделанных в проекте изменений рекомендуем при работе в среде Visual Basic установить режим автоматического сохранения всех откорректированных компонентов при запуске программы на выполнение. Для этого нужно командой Tools|Options вызвать диалоговое окно Options, открыть там вкладку Environment и в переключателе When a program starts установить позицию Save Changes. В таком режиме работы запуск программы будет выполняться несколько медленнее (скорее всего, вы этого даже не заметите), но вам будет гарантирована сохранность результатов вашего труда в случае неожиданного аварийного завершения VB при отладке приложения (что хотя и маловероятно, но все же случается). К сожалению, в среде VBA, включенной в офисные пакеты Office 97, не предусмотрено такое автоматическое сохранение проекта — за этим нужно следить самому разработчику.

10. Если при работе в среде VBA вы вносите изменения в загруженный готовый модуль (например удаляете ненужную процедуру), то сразу поменяйте его название, чтобы не путать его с исходным файлом. Аналогично, если вы меняете код процедуры, то скорректируйте ее имя и сделайте соответствующую пометку в комментарии. Храните все повторно используемые процедуры в отдельных модулях. Созданные в рамках данного приложения процедуры, которые могут пригодиться в дальнейшем, записывайте в виде отдельного файла и бережно храните.

дрей Колесов, Ольга Павлова

© 1999, Андрей Колесов, Ольга Павлова
Авторский вариант. Статья была опубликована с незначительной литературной правкой в журнале "КомпьютерПресс" N 03/99, компакт-диск.

Совет 182. Реализация функции "ожидания" в VB

Здесь мы покажем, как можно реализовать функцию ожидания в VB. Вначале поместите на форму таймер (Timer1) и установите его свойство Interval=0 и свойство Enabled=False.

Для тестирования процедуры добавьте две метки (Label1 и Label2) и командную кнопку (Command1) к форме. Напишите следующую подпрограмму и код события Timer для таймера:

```
Public Sub Wait(seconds)
    ' включение таймера
    Timer1.Enabled = True
    'установка интервала для таймера
    Me.Timer1.Interval = 1000 * seconds
    While Me.Timer1.Interval > 0
        DoEvents
    Wend
    ' выключение таймера
    Timer1.Enabled = False
End Sub

Private Sub Timer1_Timer()
    Timer1.Interval = 0
End Sub
```

Теперь можете использовать функцию Wait везде, где требуется какая-либо задержка, например:

```
Private Sub Command1_Click()
    Label1.Caption = Now
    Wait (5)
    Label2.Caption = Now
End Sub
```

Совет 183. Поддержка нумерации версий ваших VB-программ

Нумерация версий программ, созданных на VB, может быть простой, если вы воспользуетесь функцией Version Numbering при создании EXE-модуля. Для этого щелкните кнопку Options диалогового окна Make Project, а затем установите флажок Auto Increment во вкладке Make диалогового окна Project Properties.

Номер версии программы состоит из трех элементов: Major, Minor и Revision. Функция Auto Increment, если она выделена, будет автоматически увеличивать номер Revision на единицу при каждом запуске команды Make Project для конкретного проекта.

Обычно информация о версиях программы используется в форме About. Для этого просто добавьте метку с именем lblVersion и введите следующий код для вашей формы:

```
lblVersion.Caption = "Версия: " & App.Major & _
    "." & App.Minor & "." & App.Revision
```

Если для вашей программы номер Major равен 2, номер Minor — 1 и номер Revision — 12, то метка выведет на экран: "Версия: 2.1.12"

Совет 184. Создание собственного хранителя экрана

У вас никогда не было желания создать свой собственный хранитель экрана в VB? Если да, то сейчас мы вам покажем простой пример, как это можно сделать. Идея очень проста — в качестве заставки будет выдаваться форма размером с весь экран, на которую вы можете по собственному желанию нанести любые изображения.

Для начала создайте новый проект Standard EXE. Поместите на форму элемент управления Label, содержащий какой-либо текст. Затем добавьте туда элемент управления Timer, для которого установите свойство Interval как 1000 (т.е. 1 сек). Теперь введите следующий код для вашей формы:

```
Private Sub Form_Click()
    ' хранитель экрана выгружается, если щелкнуть форму
    Unload Me
End Sub

Private Sub Form_Load()
    ' не разрешается загружать несколько
    ' экземпляров хранителя экрана
    If App.PrevInstance Then Unload Me
End Sub

Private Sub Timer1_Timer()
    ' мигание метки каждую секунду
    Label1.Visible = Not (Label1.Visible)
End Sub
```

После этого установите свойство WindowState для формы как Maximized, а свойство Border Style как None. Большинство хранителей экрана занимают полный размер экрана и не имеют строки заголовка.

Выберите команду File|Make EXE File и в появившемся диалоговом окне Make Project щелкните кнопку Options. В следующем диалоговом окне Project Properties в текстовом поле Application Title введите прописными буквами строку SCRNSAVE:. (Например, мы можем назвать наше приложение SCRNSAVE:TestApp1.) При задании имени исполняемого файла не забудьте поменять расширение: оно должно быть .SCR вместо .EXE. (Для нашего примера назовите исполняемый файл как TestApp1.scr.) Щелкните OK.

Вот и все. Теперь не забудьте поместить SCR-файл в каталог \Windows\System и поменяйте хранитель экран, как это вы обычно делаете с помощью Control Panel.

Совет 185. Используйте WithEvents для добавления новых функций к элементу управления

Когда-нибудь вы можете столкнуться с тем, что у стандартных элементов управления отсутствуют какие-либо полезных функций. Для этого можно использовать традиционный способ: написать нужный код для соответствующего события каждого элемента управления. Однако в VB5 и VB6 имеется команда WithEvents, которая предоставляет простое решение с использованием классов в таких случаях.

Предположим, вы хотите вводить в текстовое окно только прописные буквы, чтобы при этом все вводимые строчные буквы автоматически преобразовывались в прописные. Кроме того, вы хотите, чтобы при попадании курсора мыши внутрь текстового окна на экране появлялась всплывающая подсказка ToolTipText, содержащая координаты курсора внутри данного текстового окна.

Создайте новый проект Standard EXE, разместите на форме четыре элемента управления TextBox с именами Text1, Text2, Text3, Text4 и добавьте модуль класса. Введите следующий код для формы Form1:

```
' общие объявления
Private clsTextBox1 As Class1
Private clsTextBox2 As Class1
Private clsTextBox3 As Class1
Private clsTextBox4 As Class1

Private Sub Form_Load()
    Set clsTextBox1 = New Class1
    Set clsTextBox1.TextBoxCtl = Text1
    Set clsTextBox2 = New Class1
    Set clsTextBox2.TextBoxCtl = Text2
    Set clsTextBox3 = New Class1
    Set clsTextBox3.TextBoxCtl = Text3
    Set clsTextBox4 = New Class1
    Set clsTextBox4.TextBoxCtl = Text4
End Sub

Private Sub Text4_KeyPress(KeyAscii As Integer)
    ' этот код выполняется перед событием KeyPress
    ' для класса Class1
    If KeyAscii = Asc("a") Then
      KeyAscii = Asc("z"): Beep
    End If
End Sub
```

Для класса Class1 введите такой код:

```
Private WithEvents txt As TextBox

Public Property Set TextBoxCtl(OutsideTextBox As TextBox)
    Set txt = OutsideTextBox
End Property

Private Sub txt_KeyPress(KeyAscii As Integer)
    ' преобразует в прописные буквы
    KeyAscii = Asc(UCase$(Chr$(KeyAscii)))
End Sub

Private Sub txt_MouseMove(Button As Integer,
    Shift As Integer, X As Single, Y As Single)
    txt.ToolTipText = "X:" & X & " Y:" & Y
End Sub
```

Используя такую программную конструкцию, вы добавите новые функциональные возможности для всех четырех текстовых полей. Но обратите внимание, что событие KeyPress для элемента управления выполняется раньше, чем происходит обращение к классу. Поэтому в поле TextBox4 вместо "a" будет вводится "z", которая потом сразу преобразуется в "Z".

Совет 186. Изменение набора изображений в элементе управления ImageList, связанном с элементом управления Toolbar

В режиме разработки проекта вам может пригодиться возможность свободно добавлять изображения в элемент управления ImageList, связанный с элементом управления Toolbar, или удалять их оттуда. И поскольку VB не позволяет изменять набор изображений в ImageList, пока он связан с панелью инструментов, мы покажем вам способ, как обойти данное ограничение.

Шаг 1. Заполнение элемента управления ImageList. Поместите элемент управления ImageList на форму. (Если данный компонент не входит в комплект инструментальных средств вашего проекта, то его можно добавить так, как показано в следующем Совете 187.) Щелкните его правой кнопкой мыши, а затем выберите команду Properties для открытия диалогового окна Property Pages. Выберите вкладку Images и щелкните кнопку Insert Picture. В диалоговом окне Select Picture найдите изображение, которое хотите добавить в элемент управления ImageList. Присвойте ему уникальное свойство Key. Повторите эти операции, пока не заполните элемент управления ImageList так, как вам хочется.

Шаг 2. Добавление кнопок к панели инструментов. Щелкните правой кнопкой мыши элемент управления Toolbar и затем выберите команду Properties. В раскрывшемся диалоговом окне Property Pages выберите вкладку Buttons. Щелкните кнопку Insert Button и в текстовом поле Key введите уникальное имя, присвоенное изображению в элементе управления ImageList. Каждая кнопка с изображением должна иметь то же свойство Key, что и соответствующее изображение в компоненте ImageList. Каждая кнопка без изображения, например, tbrSeparator или tbrPlaceholder, не должна иметь свойства Key.

Шаг 3. В событии Load для формы установите связь элементов управления ImageList и Toolbar:

```
Set ToolBar1.ImageList = ImageList1
```

Шаг 4. Присвойте изображения кнопкам на панели инструментов:

```
Dim myButton as Variant

For Each myButton in ToolBar1.Buttons
    If myButton.Key <> Empty Then
        myButton.Image = myButton.Key
        ' если значение свойство Key имеет
        ' какой-либо смысл, используйте его
        ' для описания и текста подсказки
        myButton.Description = myButton.Key
        myButton.ToolTipText = myButton.Key
    End If
Next
```

Совет 187. Загрузка элементов управления ActiveX

Для использования элементов управления ActiveX, поставляемых с VB 5.0/6.0, необходимо добавить их к комплекту инструментальных средств Toolbox.

Шаг 1. В меню Project выберите команду Components или щелкните правой кнопкой мыши панель инструментов для вызова диалогового окна Components.

Шаг 2. Элементы, приведенные в этом диалоговом окне, включают все зарегистрированные встроенные объекты, проектировщики и элементы управления ActiveX.

Шаг 3. Установите флажок, находящийся слева от имени элемента управления, который вы хотите добавить.

Шаг 4. Щелкните OK для закрытия диалогового окна Components. Теперь все выбранные элементы управления ActiveX появятся в комплекте инструментальных средств Toolbox.

Такая процедура очень проста, если вы знаете точное имя добавляемого элемента управления. Проблема возникает тогда, когда элемент, приведенный в диалоговом окне Components, содержит несколько компонентов или его название отличается от имени компонента. Здесь вам может помочь приведенная ниже таблица с перечнем э элементов управления ActiveX, поставляемые с VB 5.0/6.0:

Элемент диалогового окна Components	Имя компонента
Microsoft Windows Common Controls	TabStrip Toolbar StatusBar ProgressBar TreeView ListView ImageList Slider ImageCombo
Microsoft Windows Common Controls-2	Animation UpDown MonthView DTPicker FlatScrollBar
Microsoft Windows Common Controls-3	CoolBar
Microsoft MAPI Controls	MAPISession MAPIMessages
Microsoft Data Bound List Controls	DBList DBCombo
Microsoft DataList Controls	DataList DataCombo
Microsoft Direct Animation Media Controls	PathControl StructuredGraphicsControl SpriteControl SequencerControl
Microsoft Internet Controls	WebBrowser ShellFolderViewOC

Подробнее о новых и усовершенствованных элементах управления, поставляемых вместе VB 6.0, см. "Visual Basic 6.0 появился..."[20].

Совет 188. Использование типа Date с источником данных ADO

Первое, что приходит в голову при работе с датами в VB, — это использовать переменную типа Date. Однако в действительности такой подход оказывается неверен, если вы имеете дело с датами Null, получаемыми из источника данных ADO. Причина заключается в том, что поведение внутреннего типа данных Date отличается от типа данных adDate ADO.

Чтобы увидеть эту разницу между Date и adDate, внимательно изучите следующий код (для VB6 не забудьте установить ссылку Microsoft ActiveX Data Objects Recordset 2.0 Library, но то же самое верно и для Microsoft ActiveX Data Objects 2.0 Library):

```
Private Sub Date_handling()
    Dim lDate As Date
    Dim lDateVar As Variant
    Dim lRs As ADOR.Recordset
    '
    ' инициализация
    Set lRs = New ADOR.Recordset
    lRs.Fields.Append "MyDate", adDate, , adFldIsNullable
    lRs.Open
    lRs.AddNew
    lRs!MyDate = Date
    MsgBox lRs!MyDate
    '
    ' хранение даты
    lDate = lRs!MyDate
    lDateVar = lRs!MyDate
    '
    ' установка даты
    'lDate = Null    ' этот оператор не работает
    lDateVar = Null ' этот оператор будет работать
    '
    lRs!MyDate = lDateVar
    'lRs!Update
    lRs.Close
End Sub
```

Переменная типа String — тоже не лучший вариант, поскольку строковая переменная Null не является представлением даты Null. К сожалению, согласно установке, используемая по умолчанию в DataEnvironment в VB6, при перемещении поля Date с помощью метода "перетащи и оставь" на форму помещается элемент управления TextBox.

Таким образом, чтобы получить правильное внутреннее представление даты, следует использовать переменную типа Variant.

Свойство App.Path может использоваться для получения пути к текущему исполняемому файлу приложения. Будьте, однако, осторожны, так как при этом возможен небольшой ляп. Если приложение выполняется в корневом каталоге, то в конце пути будет добавляться обратная косая черта. Однако, если приложение выполняется в каком-либо

[20] http://www.visual.2000.ru/develop/ms-vb/cp9901/vb6-new.htm

другом каталоге, то результат не будет иметь обратной косой черты на конце. Использование следующей функции поможет решить данную проблему:

```
Public Function AppPath(sFileName As String) As String
    If Right$(App.Path, 1) = "\" Then
        AppPath = App.Path & sFileName
    Else
        AppPath = App.Path & "\" & sFileName
    End If
End Function
```

Например, AppPath("test.txt") будет правильно добавлять имя файла независимо от того, в каком каталоге находится приложение.

Совет 190. Экспорт содержимого элемента управления Grid в текстовый файл

Здесь приводится подпрограмма, которая используется для экспорта содержимого элементов управления MSGrid или MSFlexGrid в ASCII-файл неограниченного размера. В качестве разделителя вы можете задавать любой символ, который вам нравится. Кроме того, у вас есть возможность указывать символ, в который будет заключаться содержимое каждой ячейки. Используя следующий, например, вызов подпрограммы, вы заключите содержимое ячеек в двойные кавычки:

```
GridExport(grdEmployees, "c:\test.cvs", ",", Chr$(34))
```

А так выглядит сама подпрограмма:

```
Public Sub GridExport(GridToExport As Object, _
    FileName As String, Optional Delimiter As Variant, _
    Optional EncloseStrings As Variant)
    ' GridToExport — имя экспортируемого элемента
    ' управления MSGrid  или MSFlexGrid;
    ' FileName — полное имя файла, куда экспортируется
    ' содержимое сетки;
    ' Delimiter — символ-разделитель (необязательный
    ' параметр, по умолчанию используется Tab);
    ' EncloseStrings — символ, в который заключается
    ' содержимое каждой ячейки (необязательный
    ' параметр, по умолчанию ничего не используется)
    Dim iNumRows As Integer
    Dim iNumCols As Integer
    Dim iFileNumber As Integer
    '
    If IsMissing(Delimiter) Then
      Delimiter = vbTab
    End If
    If IsMissing(EncloseStrings) Then
      EncloseStrings = ""
    End If
    iFileNumber = FreeFile
    Open FileName For Output As #iFileNumber
      For iNumRows = 0 To GridToExport.Rows - 1
        GridToExport.Row = iNumRows
        For iNumCols = 0 To GridToExport.Cols - 1
          GridToExport.Col = iNumCols
          ' если это не первый столбец, то
          ' перед значением ставится разделитель
          If iNumCols > 0 Then
```

```
            Print #iFileNumber, Delimiter;
          End If
          Print #iFileNumber, EncloseStrings & _
                GridToExport.Text & EncloseStrings;
        Next iNumCols
        Print #iFileNumber, ""
      Next iNumRows
    Close #iFileNumber
End Sub
```

Андрей Колесов, Ольга Павлова

© 1999, Андрей Колесов, Ольга Павлова
Авторский вариант. Статья была опубликована с незначительной литературной правкой в журнале "КомпьютерПресс" N 06/99, компакт-диск.

Совет 191. Как определять високосный год

Те, кто думают, что високосный год — это тот, который делится без остатка на четыре, глубоко заблуждаются. Так определяется високосный год в Юлианском календаре (старый стиль). А вот в Григорианском (новый стиль) — для устранения несоответствия календарного и солнечного (астрономического) года 100-й год не считается високосным, но каждый 400-й — считается. Ошибка средней продолжительности года в этом алгоритме составляет всего 26 секунд.

Таким образом, алгоритм определения високосного года выглядит следующим образом:

```
Public Function IsLeapYear(iYear As Integer)
  ' Проверка високосного года
  If (iYear Mod 4 = 0) And _
      ((iYear Mod 100 <> 0) Or (iYear Mod 400 = 0)) Then
    IsLeapYear = True ' високосный
  Else
    IsLeapYear = False ' не високосный
  End If
End Function
```

Но можно придумать алгоритм гораздо проще. Еще раз — что такое високосный год? Правильно, тот, который имеет дату 29 февраля. А значит, работает такой алгоритм:

```
Function IsLeap (sYear As String) As Integer
  IsLeap = False
  If IsDate ("02/29/" & sYear) Then IsLeap = True
End Function
```

Однако по нашему мнению, данный алгоритм определения високосного года можно применять только к Григорианскому календарю, который был введен в действие в 1582 году. А до этого момента следует использовать правило Юлианского календаря (подробнее об этом см. "Y2K: как вести календарь"[21]).

[21] http://www.visual.2000.ru/kolesov/compress/90307lea.htm

Совет 192. Форматирование числа при выводе

Иногда бывает полезно выводить числовую информацию с фиксированным числом знаков, заполняя левые позиции нулями. Для этого можно воспользоваться следующей функцией:

```
Function PadToString(myValue, Digits) As String
  Dim Digits, MyValue
  PadToString = String(Digits - Len(myValue), "0") & myValue
End Function
```

Сделав, например, такое обращение

```
NewStr$ = PadToString(1978, 8)
```

вы получите строковую переменную 00001978. Обратите внимание, что Digits и myValue — переменные типа Variant.

03.11.2007 я получил такое письмо читателя:

Приведенный вами пример не будет работать по нескольким причинам:

1. `Function PadToString(myValue, Digits) As String`
2. `Dim Digits, MyValue`
3. переменные уже объявлены в названии функции в связи с чем компилятор при повторном объявлении будет на них ругаться.
4. При вводе строки myValue по длине большей, чем указано в Digits получаем ошибку в функции PadToString = String(Digits - Len(myValue), "0") & myValue из-за отрицательного значения Digits - Len(myValue) ну и соответственно функция ничего не вернет, если указать такие аргументы.

Вот так все работает:

```
Function PadToString(MyValue, Digits) As String
If Digits - Len(MyValue) > 0 Then
    PadToString = String(Digits - Len(MyValue), "0") & MyValue
    Else
    PadToString = MyValue
End If
    End Function
```

Совет 193. Как измерять временные интервалы

Для измерения временных интервалах между двумя произвольными моментами времени удобнее всего использовать встроенную функцию:

```
DateDiff (Interval, StartDate, EndDate[, FirstDay,[ FirstWeek]])
```

Строковая переменная Interval задает единицы измерения интервала — от секунд до года. Однако часто при тестировании скорости выполнения различных программных конструкций (BenchMark) секундной дискретности бывает недостаточно. В этом случае лучше воспользоваться еще одними внутренними системными часами Windows, которые измеряют время в "тиках" — миллисекундах с момента последнего старта или перезагрузки операционной системы. (При непрерывной работе компьютера обнуление

счетчика происходит примерно каждые 49 суток.) Чтение значений "тиков" в Win32 производится с помощью API-функции GetTickCount& (в Win16 для этого использовалась функция GetCurrentTime).

Пример вычисления временных интервалов в программе может выглядеть таким образом:

```
Private Declare Function GetTickCount& Lib "kernel32" ()

Private Sub Form_Load()
  Dim dStart As Date
  Dim lCount&, i&, strValue$, msec&
  '
  lCount = 1000000  ' счетчик циклов
  dStart = Now
  msec& = GetTickCount&
  For i = 1 To lCount
    strValue = "Петя" & "+" & "Вася"
  Next
  MsgBox "Интервал: сек = " & _
    DateDiff("s", dStart, Now) & _
    " мсек = " & GetTickCount& - msec&
End Sub
```

Однако следует иметь в виду, что вычисленный таким образом интервал не является "чистым" временем выполнения данного программного кода. В него, включается также время выполнения более приоритетных заданий операционной системы — работа других приложений, системных драйверов и пр. Для более точного определения "чистого" времени выполнения конкретного вычислительного процесса с точностью до 100 наносекунд в составе Win32 API системы Windows NT имеется функция GetProcessTime.

Совет 194. Как узнать значения кодовых таблиц

Как известно, Windows использует в своей работе две кодовые таблицы, которые мы обычно называем DOS и Windows, но формально они именуются OEM и ANSI. Довольно многие функции обработки строковых переменных зависят от значения кодовых таблиц, установленных при инсталляции системы, которые можно определить с помощью функций API:

```
Private Declare Function GetACP Lib "kernel32" () As Long
Private Declare Function GetOEMCP Lib "kernel32" () As Long

Private Sub Form_Load()
  MsgBox "ACP = " & GetACP & vbCrLf & "OEMCP = " & GetOEMCP
End Sub
```

В принципе значения кодовых таблиц можно менять уже после инсталляции Windows. Мы не советуем злоупотреблять этим, но такая коррекция может быть полезной при тестировании и анализе работы программ. Параметры ACP и OEMCP хранятся в разделе HKEY_Local_Machine/System/CurrentControlSet/Control/Nls/Codepage файла Реестра и редактируются с помощью утилиты REGEDIT.EXE. Для их активизации нужно перезагрузить Windows.

Номера таблиц, которые могут понадобиться:

OEMCP = 866 (Russian), 437 (US — default), 850 (International), 855 (Cyrilic)

ACP = 1250 (Eastern European), 1251 (Cyrilic & Russian), 1252 (US & Western European), 1200 (Unicode)

Совет 195. Как запустить Web-браузер из VB-приложения

Здесь приводится довольно простая функция, позволяющая запустить используемый по умолчанию Web-браузер из своего VB-приложения.

Введите следующий код в раздел General для модуля:

```
Declare Function ShellExecute Lib "shell32.dll" _
  Alias "ShellExecuteA" _
  (ByVal hwnd As Long, ByVal lpOperation As String, _
  ByVal lpFile As String, ByVal lpParameters As String, _
  ByVal lpDirectory As String, ByVal nShowCmd As Long) As Long

Public Const SW_SHOWNORMAL As Long = 1
Public Const SW_SHOWMAXIMIZED As Long = 3
Public Const SW_SHOWDEFAULT As Long = 10
```

Затем поместите в форму код, запускающий используемый по умолчанию Web- браузер:

```
Public Sub RunBrowser(strURL As String, iWindowStyle As Integer)
  Dim lSuccess As Long

  '— Shell to default browser
  lSuccess = ShellExecute(Me.hwnd, "Open", _
          strURL, 0&, 0&, iWindowStyle)
End Sub
```

Для запуска какого-либо Web-узла, к примеру www.visual.2000.ru, используйте такой вызов функции:

```
Call RunBrowser ("www.visual.2000.ru", SW_SHOWNORMAL)
```

Совет 196. Чтобы избежать появления ошибок, преобразовывайте значения NULL в пустые строковые переменные

При получении значений NULL из объекта Recordset могут возникать ошибки. Одним из способов избежать такой ситуации является проверка значения поля, и в том случае, если оно равно NULL, преобразование его в пустую строковую переменную или в ноль. Например,

```
If isnull(rs("Field")) then tmp="" else tmp=rs("Field")
form.textfield=tmp
```

Но еще проще использовать функцию форматирования, которая бы автоматически преобразовывала значение NULL в пустую строку. Такая функция выглядит примерно следующим образом:

```
form.textfield=format(rs("Field"))
```

Совет 197. Создание Word-документов при помощи VB-кода

Вы никогда не хотели создавать профессионального вида документы, наподобие тех, что пишутся в Microsoft Word, при помощи программного кода на VB? Следуйте приведенным ниже инструкциям, и у вас появится такая возможность.

Шаг 1. Добавьте к проекту ссылку к Microsoft Word 8.0 Object Library (команда Project|References).

Шаг 2. Введите следующий код для создания экземпляра Word и напишите какой-либо текст в новом документе:

```
Dim objWord As New Word.Application

'— Выводит Microsoft Word
objWord.Visible = True

'— Добавляет новый документ
objWord.Documents.Add

'— Вводит текст в документ
objWord.Selection.TypeText "Visual Basic!"

'— Выделяет весь текст
objWord.Selection.WholeStory

'— Изменяет размер шрифта
objWord.Selection.Font.Size = 50

Set objWord = Nothing
```

Шаг 3. Внимательно изучите Object Browser, чтобы воспользоваться другими свойствами и методами, предоставляемыми объектом Word.

Совет 198. Разбор полей строковой переменной при помощи функции SPLIT

Процедуры для проведения синтаксического разбора являются наиболее часто обновляемыми среди всех функций, манипулирующими строковыми переменными. VB6 не стал исключением, и в его состав вошла новая функция SPLIT. Она очень проста в использовании — всего одна строка кода и вы можете анализировать любую строку при помощи конкретного разделителя. Эта строка будет выглядеть следующим образом:

```
Dim strAnimals As String
Dim iCounter As Integer
Dim arrAnimals() As String

strAnimals = "Cats,Dogs,Horses,Birds"

'— Синтаксический разбор строки
arrAnimals = Split(strAnimals, ",")

'— Просмотр массива в цикле
For iCounter = LBound(arrAnimals) To UBound(arrAnimals)
  MsgBox arrAnimals(iCounter)
Next
```

Совет 199. Создайте свою собственную функцию Format

Команда Format, входящая в состав VB5, работает практически аналогично команде Print. Отличие состоит в том, что Format укорачивает выводимую строковую переменную, если количество символов форматирования превышает ее длину Для решения этой проблемы создадим свою функцию с названием FormatNum.

```
Public Function FormatNum(MyNumber As Double, FormatStr As String)

  ' Эта функция возвращает число, отформатированное как строковая
  'переменная, содержащая требуемое минимальное количество символов
  '
  ' MyNumber — Используйте CDbl(MyNumber) при вызове функции, чтобы
  ' избежать появления ошибки о несовпадении типов
  '
  FormatNum = Format(MyNumber, FormatStr)
  If Len(FormatNum) < Len(FormatStr) Then
    FormatNum = Space(Len(FormatStr) - Len(FormatNum)) & FormatNum
  End If
End Function
```

Использование новой функции Format проиллюстрируем на таком примере:

```
Private Sub Form_Load()
Dim MyVar, FormatStr$, f1$, f2$
MyVar = 12.1
FormatStr$ = "####.##"
f1$ = Format$(MyVar, FormatStr$)
f2$ = FormatNum(MyVar, FormatStr$)

MsgBox f1$ & "  " & Len(f1$) & vbCrLf & f2$ & "  " & Len(f2$)
End Sub
```

Совет 200. Как осуществить выход из Windows с помощью VB

Вам никогда не хотелось сделать так, чтобы ваши приложения могли автоматически выполнить операцию выхода из Windows? В этом нет ничего сложного, просто нужно использовать вызов соответствующей API-функции. Для этого добавьте следующее описание функции и константы в BAS-модуль:

```
Declare Function ExitWindowsEx& Lib _
  "user32" (ByVal uFlags&, ByVal wReserved&)

' константы, необходимые для выхода из Windows
Global Const EWX_FORCE = 4      ' закрытие неактивных приложений
Global Const EWX_LOGOFF = 0     ' выход из системы
Global Const EWX_REBOOT = 2     ' перезагрузка
Global Const EWX_SHUTDOWN = 1   ' закрытие системы
```

Для выключения Windows используйте такой вызов функции:

```
' выключает компьютер
lresult = ExitWindowsEx(EWX_SHUTDOWN, 0&)
```

Для выполнения других операций замените первый параметр при вызове функции ExitWindowsEx на соответствующую константу.

Совет 201. Для ускорения процесса загрузки изменяйте базовые адреса компонентов типа in-process

При загрузке собственного компонента типа in-process в ходе выполнения VB-приложения, этот компонент размещается, начиная с некоторого базового адреса памяти.

Как можно поменять этот адрес для своего компонента? Для этого откройте диалоговое окно Project Properties и выберите вкладку Compile. Адрес вводится в поле DLL Base Address в виде десятичного или шестнадцатиричного целого числа без знака. По умолчанию используется значение &H11000000 (285,212,672). Если вы забудете изменить его, ваш компонент вступит в противоречие с любым другим компонентом типа in-process, скомпилированным с учетом используемого по умолчанию адреса. Поэтому рекомендуем вам задавать какой-либо отличный от него адрес.

Выбирайте базовый адрес в диапазоне между 16 Мб (16,777,216 или &H1000000) и 2 Гб (2,147,483,648 или &H80000000). При этом он должен быть кратным 64 Кб. Область памяти, используемая вашим компонентом, начинается с исходного базового адреса и имеет размер скомпилированного файла, округленного до следующего числа, кратного 64 Кб.

Ваша программа не может превышать 2 Гб, поэтому максимальный базовый адрес фактически равен 2 Гб минус область памяти, занимаемая созданным компонентом. Исполняемые файлы обычно будут загружаться по логическому адресу в 4 Мб. Область памяти меньше 4 Мб резервируется для Windows 95, а области свыше 2 Гб — для Windows 95 и Windows NT.

Чтобы убедиться, что все ваши компоненты имеют различные базовые адреса, необходимо вести их учет. Для этого полезно разработать собственный инструмент, который создавал бы новые уникальные базовые адреса. Таким образом ваши компоненты не вступали бы в противоречие друг с другом.

Совет 202. Зачем нужен параметр Alias в операторе Declare.

Издавна (еще в версиях для DOS) в операторе Declare использовался ключ Alias для переопределения имени вызываемой процедуры. Конструкция

```
Declare Function vbname Lib libname [Alias aliasname]
```

означает, что к процедуре, записанной в библиотеке libname под именем aliasname, в VB-программе обращаются под именем vbname. Обычно это применяется в двух случаях:

1. Когда имя aliasname просто недоступно в VB. В частности, при обращении к функциям, которые имеют (по принятым в C правилам) в начале названия символ подчеркивания, например _lOpen. Или когда внешняя функция использует имя, которое совпадает с зарезервированным ключевым словом VB (скажем SetFocus).
2. Когда подразумевается, что VB-программа может работать с двумя версиями одной и той же (с точки зрения функциональности) процедуры. В этом случае гораздо проще исправить один оператор Declare:
3. `Declare Function vbname Lib libname Alias aliasname1`

   на

```
Declare Function vbname Lib libname Alias aliasname2
```

чем менять во всей программе имя процедуры vbname.

Второй вариант особенно часто используется для обеспечения преемственности кода при переходе от Win16 к Win32 — имена многих API-функций при этом поменялись. В этой связи нужно специально выделить часто встречающийся случай, когда к привычному имени, типа SomeFunction (Win16), прибавился суффикс A и получилось SomeFunctionA (Win32). Почему так случилось, расскажет следующий совет.

Совет 203. Помните: VB использует кодировку ANSI при обращении к API-функциям.

Дело в том, что Windows 3.x (Win16) использовала только один формат хранения символьных данных (строковых переменных) — ANSI (набор однобайтовых символов). Система Win32 представлена двумя вариантами — Windows 9x и Windows NT, которые применяют разные внутренние форматы: соответственно ASNI и Unicode (набор целочисленных, двубайтовых символов). В результате, одна API-функция в Win32, например GetWindowText, фактически реализована в двух вариантах, с добавлением к имени суффикса A (символы передаются в кодировке ANSI) и W (Wide = Unicode): GetWindowTextA и GetWindowTextW. При этом следует иметь в виду, что физически в Windows 9x и Windows NT используется разная реализация таких функций.

Windows 9x допускает использование только ANSI-версий процедур и совсем не поддерживает Unicode. Но при этом она включает оба варианта функций, второй из которых на самом деле не работает. Представляется вполне логичным, если бы обращение, скажем к GetWindowTextW, вызывало бы сообщение об отсутствии такой функции. Однако в реальности данная функция не выполняет никаких действий и возвращает в вызывающую программу нулевое значение, которое можно интерпретировать как ошибку. Однако если программный код:

```
ItemHeight& = SendMessageA(.hWnd, LB_GETITEMHEIGHT, 0, 0)
NewIndex& = ListHeight& \ ItemHeight&
```

работает нормально, то замена SendMessageA на SendMessageW вызовет программную ошибку "деление на ноль".

В Windows NT оба варианта функций являются рабочими, что позволяет использовать кодировку и ANSI, и Unicode. В первом случае нужно обращаться к A-процедуре, которая преобразовывает символьный код из ANSI в Unicode и передает управление на собственно обработку в W-процедуру. Во втором случае нужно обращаться напрямую в W-процедуру.

32-разрядные версии Visual Basic используют Unicode для внутреннего хранения строковых переменных. Однако при обращении к API- или DLL-функциям, описанным с помощью оператора Declare, производится автоматическое преобразование символьных данных в ANSI (а потом — обратно). Таким образом, при работе с VB нужно обращаться к ANSI-варианту API-функций как в Windows 9x, так и Windows NT.

В принципе имя, типа SendMessageA, можно использовать непосредственно в программе. Но общепринятым считается применение имени функции без суффикса, для чего применяется параметр Alias:

```
Declare Function SendMessage ... Alias "SendMessageA"
```

Во-первых, это обеспечивает определенную совместимость имен с API Win16, а во-вторых, подчеркивает, что используется один из вариантов функции SendMessage.

Андрей Колесов, Ольга Павлова

© 199x, Андрей Колесов, Ольга Павлова
Авторский вариант. Статья была опубликована с незначительной литературной правкой в журнале "КомпьютерПресс" N 10/99, компакт-диск.

Совет 204. Как нарисовать рамку на форме без помощи элемента управления Frame

Функция DrawEdge, входящая в состав Win32 API, позволяет достичь очень интересных эффектов. Используя константы EDGE_ вы можете задать различные типы границ рамки, благодаря чему она будет выглядеть утопленной или приподнятой над формой. А константы BF_ определяют границы рамки, которые следует рисовать (например, с помощью BF_BOTTOM вы можете нарисовать только нижнюю границу рамки):

```
Private Declare Function DrawEdge Lib "user32" _
  (ByVal hDC As Long, qrc As RECT, ByVal edge As Long, _
  ByVal grfFlags As Long) As Long
Private Declare Function GetClientRect Lib "user32" _
  (ByVal hWnd As Long, lpRect As RECT) As Long
Private Type RECT
  Left As Long
  Top As Long
  Right As Long
  Bottom As Long
End Type
Const BDR_INNER = &HC
Const BDR_OUTER = &H3
Const BDR_RAISED = &H5
Const BDR_RAISEDINNER = &H4
Const BDR_RAISEDOUTER = &H1
Const BDR_SUNKEN = &HA
Const BDR_SUNKENINNER = &H8
Const BDR_SUNKENOUTER = &H2
Const BF_RIGHT = &H4
Const BF_LEFT = &H1
Const BF_TOP = &H2
Const BF_BOTTOM = &H8
Const EDGE_BUMP = (BDR_RAISEDOUTER Or BDR_SUNKENINNER)
Const EDGE_ETCHED = (BDR_SUNKENOUTER Or BDR_RAISEDINNER)
Const EDGE_RAISED = (BDR_RAISEDOUTER Or BDR_RAISEDINNER)
Const EDGE_SUNKEN = (BDR_SUNKENOUTER Or BDR_SUNKENINNER)
Const BF_RECT = (BF_LEFT Or BF_RIGHT Or BF_TOP Or BF_BOTTOM)
```

В событии Form_Paint введите следующий код, который говорит о том, что вы хотите нарисовать прямоугольник, который приподнят над формой:

```
Private Sub Form_Paint()
  Static Tmp As RECT
  Static TmpL As Long
  TmpL = GetClientRect(hWnd, Tmp)
```

```
  TmpL = DrawEdge(hDC, Tmp, EDGE_RAISED, BF_RECT)
End Sub
```

Совет 205. Как организовать просмотр каталогов

Здесь приводится простой программный код, с помощью которого вы можете вывести на экран окно просмотра каталогов. Благодаря этому пользователь имеет возможность выбрать необходимый ему каталог:

```
Private Type BrowseInfo
  hWndOwner As Long
  pIDLRoot As Long
  pszDisplayName As Long
  lpszTitle As Long
  ulFlags As Long
  lpfnCallback As Long
  lParam As Long
  iImage As Long
End Type
'
' просмотр каталогов
'
' для выбора каталога,
' чтобы начать поиск документа
Private Const BIF_RETURNONLYFSDIRS = &H1
' для запуска команды Find
Private Const BIF_DONTGOBELOWDOMAIN = &H2
Private Const BIF_STATUSTEXT = &H4
Private Const BIF_RETURNFSANCESTORS = &H8
' просмотр компьютеров
Private Const BIF_BROWSEFORCOMPUTER = &H1000
' просмотр принтеров
Private Const BIF_BROWSEFORPRINTER = &H2000
' просмотр всего
Private Const BIF_BROWSEINCLUDEFILES = &H4000
'
Private Const MAX_PATH = 260
Private Declare Sub CoTaskMemFree Lib "ole32.dll" _
  (ByVal hMem As Long)
Private Declare Function lstrcat Lib "kernel32" Alias "lstrcatA" _
  (ByVal lpString1 As String, ByVal lpString2 As String) As Long
Private Declare Function SHBrowseForFolder Lib "shell32" _
  (lpbi As BrowseInfo) As Long
Private Declare Function SHGetPathFromIDList Lib "shell32" _
  (ByVal pidList As Long, ByVal lpBuffer As String) As Long

Public Function BrowseForFolder(hWndOwner As Long, _
  sPrompt As String) As String
  '=================================================
  ' Открывает системное диалоговое окно для просмотра каталогов
  '=================================================
  Dim iNull As Integer
  Dim lpIDList As Long
  Dim lResult As Long
  Dim sPath As String
  Dim udtBI As BrowseInfo
  '
  With udtBI .hWndOwner = hWndOwner
    .lpszTitle = lstrcat(sPrompt, "")
    .ulFlags = BIF_RETURNONLYFSDIRS
  End With
  lpIDList = SHBrowseForFolder(udtBI)
```

```
  If lpIDList Then
    sPath = String$(MAX_PATH, 0)
    lResult = SHGetPathFromIDList(lpIDList, sPath)
    Call CoTaskMemFree(lpIDList)
    iNull = InStr(sPath, vbNullChar)
    If iNull Then sPath = Left$(sPath, iNull - 1)
    End If
  End If
  BrowseForFolder = sPath
End Function

Private Sub Form_Click()
  Dim MyStr As String
  MyStr = BrowseForFolder(hWnd, "Привет всем!")
End Sub
```

Совет 206. Удаление всех выделенных элементов в списке

Если у вас есть необходимость удалить сразу несколько элементов из списка, поддерживающего режим MultiSelect, воспользуйтесь следующей простой программой. Предположим, у нас есть окно списка, состоящее из пяти элементов: Элемент1, Элемент2, Элемент3, Элемент4 и Элемент5. Установим свойство MultiSelect элемента управления ListBox как Extended или Simple, а затем введем такой код:

```
Private Sub cmdDeleteListItems_Click()
  Dim i As Integer
  For i = List1.ListCount - 1 To 0 Step -1
    If List1.Selected(i) Then List1.RemoveItem i
  Next i
End Sub
```

Поместим на форму командную кнопку с именем "Удалить" и напишем для нее следующее:

```
Private Sub Command1_Click()
  Call cmdDeleteListItems_Click
End Sub
```

Теперь запустим нашу программу, выделим три элемента (Элемент1, Элемент3 и Элемент5), а затем щелкнем кнопку "Удалить". И это все!

Совет 207. Как сделать горизонтальную линейку прокрутки в элементе управления RichTextBox

По умолчанию, когда вы помещаете элемент управления RichTextBox на форму, VB устанавливает свойство RightMargin как 0. Это означает, что вводимый пользователем текст целиком располагается внутри текстового окна. Для вывода горизонтальной линейки прокрутки необходимо, чтобы значение свойства RightMargin было больше, чем ширина текстового окна. Иначе, даже если установить свойство ScrollBars как 1-rtfHorizontal, RichTextBox не выведет линейку прокрутки.

Например, поместите на форму элемент управления RichTextBox, имеющий ширину 3200. Затем установите свойство RightMargin как 3300, а свойство ScrollBars — как 1-rtfHorizontal. Запустите проект на выполнение и начните вводить текст. Когда вы достигнете границы текстового окна, VB выведет горизонтальную линейку прокрутки.

Совет 208. Добавление новой строки к тексту в элементе управления TextBox

Иногда вам может понадобиться добавить дополнительную информацию к уже существующему тексту в многострочном элементе управления TextBox (свойство MultiLine установлено как True). Предположим, что вы хотите добавить строку такого вида: "Обновление:" плюс текущая дата. Для этого можно воспользоваться свойствами SelStart и SelText. Как вы уже, вероятно, знаете, свойство SelStart возвращает или устанавливает начало выделения, свойство SelText — фактически выделенный текст. Если текст не содержит никакого выделения, оба свойства возвращают точку вставки (insertion point). Поэтому, чтобы вставить новую строку текста в многострочное текстовое окно, используйте подобный код:

```
Dim strNewText As String
With Text1
  strNewText = "Обновление: " & Date
  .SelStart = Len(.Text)
  .SelText = vbNewLine & strNewText
End With
```

Этот код передвигает точку вставки к конец любого текста в элементе управления Text1, а затем вставляет новую строку, содержащую дополнительную информацию.

Совет 209. Как заставить VB 6.0 открывать окно кода в "развернутом" режиме

VB 5.0 всегда сохранял установки интегрированной среды разработки (IDE), заданные вами во время последней сессии. Так, он помнил, с какими окнами вы предпочитаете работать — с нормальными или "развернутыми". К сожалению, VB 6.0 этого не делает — он всегда открывает окна Code и Object в нормальном режиме. Эту проблему можно решить, проведя небольшие изменения в Windows Registry, так что IDE будет открывать эти окна в "развернутом" режиме. Однако они будут ВСЕГДА расширены до размера экрана — VB 6.0 по-прежнему не сможет хранить установки для IDE между сессиями.

При работе с Регистром требуется соблюдать особую осторожность, поэтому, прежде чем приступить к внесению в него изменений, сделайте резервную копию, так чтобы можно было восстановить Регистр в случае каких-либо сбоев.

Итак, чтобы заставить VB 6.0 открывать окно Code или Object в "развернутом" режиме, вы должны добавить новую величину MDIMaximized к следующему ключу Registry: HKEY_CURRENT_USER/Software/Microsoft/Visual Basic/6.0/MDIMaximized = "1" Для этого в Windows щелкните кнопку Start и выберите команду Run. Введите RegEdit в диалоговом окне Run, затем щелкните OK. Тогда Windows выведет на экран системный Регистр, в котором найдите папку VB 6.0. После этого щелкните правой кнопкой мыши в любом месте на правой панели и выберите New|String Value из контекстного меню. Введите MDIMaximized в качестве имени и нажмите клавишу Enter. Теперь щелкните правой кнопкой мыши элемент MDIMaximized и выберите Modify из контекстного меню. И наконец, в диалоговом окне Edit String введите 1 в качестве новой величины и щелкните OK. Когда вы это сделаете, Windows присвоит введенную вами величину элементу MDIMaximized. Вот и все! Теперь закройте Регистр и откройте окно Code или Object в любом из проектов VB 6.0. IDE выведет эти окна в "развернутом" режиме.

Совет 210. Специальные символы для объявления типа переменной

Помимо того, что вы можете объявлять тип переменной в явном виде, VB позволяет делать это с помощью специальных символов. Например, вместо использования:

```
Dim MyString As String
```

вы можете просто написать:

```
Dim MyString$
```

Вот полный перечень типов данных и соответствующих им символов:

```
String   — $
Integer  — %
Long     — &
Single   — !
Double   — #
Currency — @
```

Однако следует соблюдать осторожность при использовании этих символов, поскольку они снижают читаемость вашего кода.

Совет 211. Создание объекта Excel в VB

В совете 197 (КомпьютерПресс 6'99, компакт-диск) мы рассказали о том, как создать объект Word в VB. Теперь мы покажем, как выполнить ту же самую процедуру для Excel.

В первую очередь добавьте к VB-приложению ссылку к Microsoft Excel 8.0 Object Library (команда Project|References).

Затем введите следующий код для создания экземпляра Excel:

```
Dim objExcel As New Excel.Application

' Выводит Microsoft Excel
objExcel.Visible = True

' Открывает новую рабочую книгу
objExcel.Workbooks.Add

' Вводит текст в ячейки таблицы
objExcel.ActiveCell(1, 1) = "Столбец 1"
objExcel.ActiveCell(1, 3) = "Столбец 3"

' Освобождает объектную переменную
Set objExcel = Nothing
```

И наконец, внимательно изучите Object Browser в VB для получения информации о других свойствах и методах объекта Excel.

Совет 212. Пусть функция DateDiff разбирается с датами

Если вам требуется определить, находятся ли две даты в одном и том же месяце, то первое, что может прийти на ум — это использовать функцию Month для каждой даты, а

затем сравнить два полученных числа. Однако в таком случае вы получите, что даты 1/1/2000 и 1/1/1999 равны. Поэтому следует использовать функцию DateDiff, которая может выглядеть примерно так:

```
DateDiff("m", Date1, Date2)
```

Здесь функция DateDiff вычисляет разницу в календарных месяцах между двумя датами (аргумент "m"). Если она возвращает 0, то обе даты находятся в одном и том же месяце. Для того, чтобы использовать эту функцию в своем приложении, вы можете использовать подобную программную конструкцию:

```
If DateDiff("m", Date1, Date2) Then
  ' Месяца отличаются
Else
  ' Один и тот же месяц
End If
```

Совет 213. Для аварийного прерывания программы используйте Ctrl+Pause

При отладке программы в среде VB для ее прерывания или аварийного завершения можно применять соответственно команды Break или End из меню Run. Однако они срабатывают только в момент ожидания какого-либо внешнего события на диалоговой форме. Если же программа выполняет какой-то программный код (например, обработку данных в цикле) или ожидает реакции пользователя после вывода окна сообщения (Message Box), то аварийно прервать или завершить ее с помощью этих команд не удастся.

Однако решить такую проблему просто — для аварийного прерывания программы нажмите комбинацию клавиш Ctrl+Pause, которая сразу переведет ваше приложение в режим Break, а уже потом используйте команды среды Continue (продолжить выполнение) или End (завершить).

Совет 214. Как увидеть начало выделенного текста

Большинство профессиональных коммерческих приложений обрабатывают диалоговые окна, содержащие поля текста, следующим образом. Когда пользователь переходит к полю ввода текста с помощью клавиши Tab или быстрой клавиши (комбинации Alt с какой-либо другой клавишей), он полностью выделяет весь текст, содержащийся в этом поле. Затем он вводит новый текст, который заменяет собой содержимое всего поле. В то же время, если он просто щелкнет мышью текст, содержащийся в поле ввода, то никакого выделения не произойдет — туда только перейдет фокус.

В документации VB Knowledge Base рассказывается, как это можно сделать с помощью API-функции GetKeyState. Однако данная техника имеет некоторое неудобство в тех случаях, когда длина текста превышает ширину поля. Пользователь тогда видит только конец выделенного текста, что не очень удобно, т.к. не всегда можно определить, о чем там идет речь.

Использование функции GetKeyState вместе с оператором SendKeys и методом TextWidth позволяет создать комбинированное решение, когда клавиша Tab или быстрая клавиша выделяет весь текст, содержащийся в поле ввода, но при этом пользователь видит начало текста, а не его конец.

Вначале опишите API-функцию GetKeyState и создайте подпрограмму SelectWholeText:

```
Option Explicit
 #If Win16 Then
   Private Declare Function GetKeyState _
     Lib "User" (ByVal iVirtKey As Integer) As Integer
 #Else
   Private Declare Function GetKeyState _
     Lib "User32" (ByVal iVirtKey As Long) As Integer
 #End If
' vbTab ' то же, что и Chr$(&H9) — символьная константа
' vbKeyTab ' то же, что и десятичная 9 — числовая константа
' vbKeyMenu ' то же, что и десятичная 18 (клавиша Alt) —
' числовая константа

Private Sub SelectWholeText(Ctl As Control)
  ' Обратите внимание на разницу в использовании
  ' vbTab (символ) и vbKeyTab(число). В первом случае
  ' необходимо добавить Asc(), чтобы получить число в
  ' качестве аргумента.
  ' Использование ключевого слова With позволяет
  ' упростить  дальнейшую модификацию программы —
  ' например, если будем менять имя параметра.
  With Ctl
    If (GetKeyState(vbKeyTab) < 0) Or _
      (GetKeyState(vbKeyMenu) < 0) Then
      ' Для выделения всего текста используется
      ' клавиша Tab или быстрая клавиша. В случае
      ' если длина строки превышает ширину поля,
      ' использование оператора SendKeys
      ' позволяет увидеть начало выделенного текста.
      ' Метод TextWidth определяет, какую длину
      ' будет иметь строка, если ее выводить на форме.
      If TextWidth(.Text) > .Width Then
        SendKeys "{End}", True
        SendKeys "+{Home}", True
      Else
        .SelStart = 0
        .SelLength = Len(.Text)
      End If
    Else
      .SelLength = 0
    End If
  End With
End Sub
```

Затем вызовите созданную подпрограмму из события GotFocus любого поля текста:

```
Private Sub Text1_GotFocus()
  SelectWholeText Text1
End Sub
```

Совет 215. Как прочитать серийный номер диска

Новая библиотека Microsoft Scripting Runtime содержит иерархию FileSystemObject, состоящую из нескольких объектов, которые позволяют получать информацию о дисках, папках и файлах. Например, вы можете получить серийный номер диска с помощью такого кода:

```
' получаем серийный номер диска c:
Dim fso As New Scripting.FileSystemObject
Dim dr As Scripting.Drive
' получаем ссылку к объекту Drive
```

```
Set dr = fso.GetDrive("c")
Print Hex$(dr.SerialNumber)
```

А используя свойство FreeSpace объекта Drive можно также проверить, достаточно ли у вас свободного места на диске:

```
Print "На диске C есть " & dr.FreeSpace & " свободных байт"
```

Более подробно об этом можно прочитать в Справке VB в разделах Dictionary и FileSystemObject.

Совет 216. Применение элемента управления Label в качестве разделителя

Для создания приложения в стиле Windows Explorer можно использовать элемент управления Label в качестве разделителя между двумя другими элементами управления, такими как ListView и TreeView.

Вначале разместим на форме эти три компонента, а затем введем следующий код:

```
Option Explicit
Private mbResizing As Boolean
' нажата ли левая кнопка мыши

Private Sub Form_Load()
  TreeView1.Move 0, 0, Me.ScaleWidth / 3, Me.ScaleHeight
  ListView1.Move (Me.ScaleWidth / 3) + 50, 0, _
    (Me.ScaleWidth * 2 / 3) - 50, Me.ScaleHeight
  Label1.Move Me.ScaleWidth / 3, 0, 100, Me.ScaleHeight
  Label1.MousePointer = vbSizeWE
End Sub

Private Sub Label1_MouseDown(Button As Integer, _
  Shift As Integer, X As Single, Y As Single)
  If Button = vbLeftButton Then mbResizing = True
End Sub

Private Sub Label1_MouseMove(Button As Integer, _
  Shift As Integer, X As Single, Y As Single)
  ' изменение размеров элементов управления
  ' при нажатой левой кнопке мыши
  If mbResizing Then
    Dim nX As Single
    nX = Label1.Left + X
    If nX < 500 Then Exit Sub
    If nX > Me.ScaleWidth - 500 Then Exit Sub
    TreeView1.Width = nX
    ListView1.Left = nX + 50
    ListView1.Width = Me.ScaleWidth - nX - 50
    Label1.Left = nX
  End If
End Sub

Private Sub Label1_MouseUp(Button As Integer, _
  Shift As Integer, X As Single, Y As Single)
  mbResizing = False
End Sub
```

Андрей Колесов, Ольга Павлова

© 1999, Андрей Колесов, Ольга Павлова
Авторский вариант. Статья была опубликована с незначительной литературной правкой в журнале "КомпьютерПресс" N 11/99 (с.180-182) и N 12 (с.176-177).

Совет 217a. Как передавать данные между формами

Читатель прислал нам такой вопрос:

Пишем программу на VB4/16, и не можем передать переменную или массив из одной формы в другую. Если Вас не затруднит, помогите советом.

Наш ответ таков:

Конечно, было проще ответить, рассмотрев конкретную конструкцию, которую пытаются реализовать читатели. Но лучше попробовать разобраться в принципах информационного взаимодействия двух форм.

1. Передача параметров через свойства объектов

Для этого создадим две формы Form1 и Form2, на каждой из которых разместим текстовое поле и командную кнопку. При такой последовательности создания форм в качестве стартовой формы приложения будет выбрана Form1. В VB4 этот параметр корректируется командой Tools|Options|Project|Startup Form, в VB5/6 — Project|Project Properties|General|Startup Object. Для элементов Form2 установим имена элементов управления как Text2 и Command2 соответственно (чтобы отличать зрительно объекты по их именам), а также Text2.Text = "Text2". Для кнопок Command1 и Command2 установим соответственно такие свойства Caption: "Запустить Form2" и "Вернуть управление в Form1". И напишем для них следующий программный код:

```
Private Sub Command1_Click()
  Form2.Show 1 ' вывод Form2 в модальном режиме
End Sub
```

для формы Form2 (для внешнего элемента управления Text1 обязательно указание имени формы Form1):

```
Private Sub Form_Activate()
  ' установка при запуске формы
  Text2.Text = Form1.Text1.Text
  Form1.Text1.Text = "" 'очистить поле
End Sub
Private Sub Command2_Click()
  ' Возврат в Form1
  ' переписываем содержимое Text2 в Text1
  Form1.Text1.Text = Text2.Text
  Form2.Hide ' закрыть форму
End Sub
```

Обратите внимание, что форма Form2 запускается в модальном режиме (значение параметра равно 1). Это означает, что фокус приложения переходит в Form2 —

родительская форма Form1 остается видимой, но недоступной для каких-либо операций. Запустите приложение и нажмите кнопку "Запустите Form2" — появится форма Form2, при этом в поле Text2 будет записано содержимое Text1. (Само поле Text1 при этом очистится). Теперь нажмите "Вернуть управление в Form1" — управление вернется в Form1 и в поле Text1 будет установлено содержимое Text2. (Вы можете сами устанавливать содержимое полей, чтобы убедиться в правильности обмена информацией.)

Таким образом мы видим, как происходит передача информации от одной формы к другой в виде установки или чтения свойств элеменов управления. Все формы и находящиеся на них элементы управления являются глобальными: их свойства и методы доступны из процедур других модулей (в том числе BAS-модулей). Вот как выглядит полное имя элемента управления:

```
[Имя_Формы.]Имя_Элемента_Управления
```

При обращении к внутренним элементам управления (внутри данной формы) можно указывать сокращенное имя объекта (что мы и делаем), а при работе с внешними — полное, которое начинается с имени формы.

2. Передача данных через параметры процедур

Продолжим наши эксперименты. Создадим на форме Form1 еще одну командную кнопку, для которой установим свойства Name = Command11 и Caption = "Второй вариант запуска", и напишем для нее такой код:

```
Private Sub Command11_Click()
  ' второй вариант запуска
  Call Form2.Text2_KeyPress(65)
End Sub
```

Для поля Text2 на форме Form2 напишем следующую процедуру:

```
Public Sub Text2_KeyPress(KeyAscii As Integer)
  Show  ' запуск формы Form2 в немодальном режиме
  Text2.Text = "Значение = " & KeyAscii
End Sub
```

Запустите на выполнение приложение, нажмите на форме Form1 кнопку Command21 и вы увидите картину, изображенную на рис. 1.

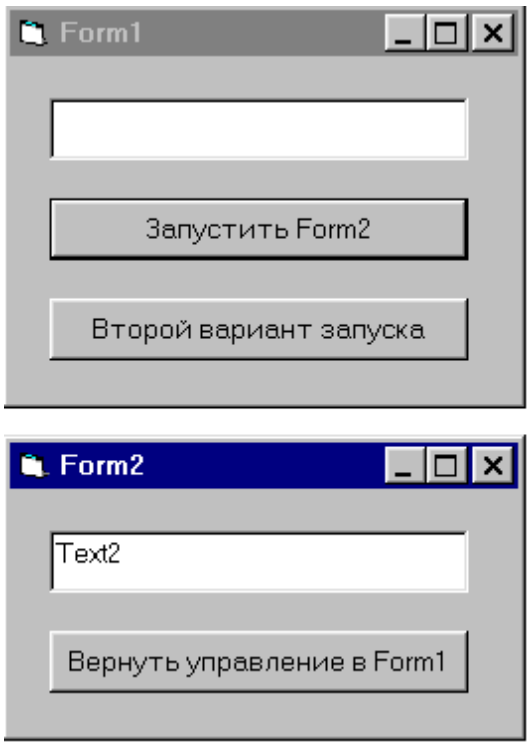


Рис. 1

В этом примере мы обратились из формы Form1 напрямую к процедуре Form2 (в данном случае к Text2_KeyPress) и даже передали в нее информацию в виде параметра процедуры. Но обратите внимание, что выше мы установили для событийной процедуры Text2_KeyPress тип Public, вместо Private по умолчанию.

Очевидно, что такое использование встроенной событийной процедуры является довольно искусственным и ограниченным по возможностям — мы можем использовать только весьма простые, предопределенные варианты передачи параметров. К тому же можно столкнуться с тем, что эти процедуры неожиданно выполняют еще какие-то встроенные операции, которые нам тут не нужны.

Гораздо лучше для реализации второго варианта создать в Form2 простую процедуру, например с именем Start3 (но обязательно типа Public):

```
Public Sub Start3(StartName$)
  Show 'запуск формы Form2
  Text2.Text = StartName$
End Sub
```

а в процедуре Command11_Click (форма Form1) записать такой код:

```
Private Sub Command11_Click()
  Call Form2.Start3("Третий вариант запуска")
End Sub
```

Очевидно, что в этом случае мы может реализовать любой вариант передачи параметров из Form1 в Form2, в том числе и массивы.

Совсем другой вопрос — что мы будем делать с передаваемыми параметрами? В приведенном здесь примере мы их сразу использовали для формирования свойств

объектов. А можно переписать эти данные в глобальные переменные на уровне данного модуля (формы), которые затем передать в другие процедуры.

3. Передача данных через глобальные переменные

Еще один вариант передачи данных между модулями — использование глобальных переменных, доступ к которым разрешен из любых процедур приложения. Однако нужно иметь в виду, что в VB такие переменные можно создавать только в модуле кода BAS. Поэтому если вы хотите, чтобы обе созданные нами формы работали с общими данными, то нужно создать BAS-модуль и в нем описать соответствующие глобальные переменные:

```
Global MyArray$(), MyVar As Integer

В форме Form1 создадим еще одну командную кнопку Command12 и запишем для нее
такой код:

%
Private Sub Command12_Click()
  ' передача данных через глобальные переменные
  ReDim MyArray$(200)
  MyArray$(1) = "Элемент1"
  MyVar = 1001
  Call Form2.Start4
End Sub
```

В форме Form2 сформируем еще одну процедуру:

```
Public Sub Start4()
  Form1.Text1.Text = MyArray$(1)
  Text2.Text = MyVar
  Show
  MsgBox UBound(MyArray$)
End Sub
```

Запустите приложение на выполнение и убедитесь, что при нажатии кнопки Command12 передача данных в Form2 выполняется правильно. Обратите внимание, что в модуле формы Form1 мы не только устанавливаем значения глобальных переменных, но и можем изменять размер массивов, объявленных динамическими.

Совет 217. Использование функции Split для вычисления количества подстрок

Как мы уже писали в Совете 198 ("КомпьютерПресс" 06'99, компакт-диск), в состав VB6 вошла новая функция, предназначенная для проведения синтаксического разбора строковых переменных. Для этого вы просто указываете конкретный разделитель внутри строки, а VB создает одномерный массив, состоящий из подстрок.

Однако функция Split обладает еще одной возможностью — вы можете использовать ее для быстрого подсчета количества подстрок внутри длинной строки. Предположим, что вы хотите определить, сколько раз слово "лес" встречается в строковой переменной "Мы пошли в лес собирать лесные ягоды". Для этого сначала проведите синтаксический разбор данной строки, а затем с помощью функции UBound подсчитайте количество элементов в полученном массиве, как это видно из следующего примера:

```
strTungTied = "Мы пошли в лес собирать лесные ягоды"
```

```
aryWood = Split(strTungTied, "лес")
MsgBox "Слово 'Лес' встречается в " & vbCrLf & _
        strTungTied & "'" & vbCrLf & _
        UBound(aryWood) & " раз(а)."
```

Теперь запустите этот код на выполнение, и VB сообщит вам, что слово "лес" встречается в заданной нами фразе два раза.

Совет 218. Будьте осторожны при использовании методов Delete объекта FileSystemObject в VB

Как вы уже, вероятно, знаете, что объектная библиотека Microsoft Scripting Runtime Library упрощает управление файлами и папками. Так, методы DeleteFolder и DeleteFile позволяют удалять любые ненужные элементы. Однако при их использовании необходимо быть особенно внимательными: в случае нахождения ошибки они прекращают свое выполнение и не совершают отказа от изменений, сделанных к этому моменту времени. Например, если метод DeleteFolder удаляет две папки из десяти, а затем возникает какая-либо ошибка, то VB прерывает выполнение и удаленными оказываются только первых два элемента. Здесь следует также помнить и том, что некоторые другие методы объекта FileSystemObject работают аналогичным образом.

Совет 219. Заменяйте длинные строки, содержащие путь, на объектные переменные (продолжение Совета 218)

Большинство методов, управляющих дисководами, папками или файлами в объектной библиотеке Microsoft Scripting Runtime Library, требуют задания аргумента filespec — полного пути, представленного в виде строковой переменной. Однако поскольку свойство Path используется по умолчанию для этих трех типов элементов, вы можете применять объектные переменные вместо строк, содержащих пути. Например, чтобы скопировать одну папку в другую с помощью объектных переменных, напишите следующее:

```
fldr1.Copy fldr2, False
```

или

```
MyFSO.CopyFile file1, file2, False
```

Совет 220. Для ограничения размеров VB-формы пользуйтесь элементом управления ActiveX

При создании VB-формы часто бывает необходимо запретить пользователю увеличивать ее за пределы каких-либо установленных размеров. Для этого можно использовать событие Resize формы и проверить, не превышают ли ее текущие размеры заданное значение (в твипах). Если это так, то VB уменьшает форму и устанавливает ее размеры в соответствии с максимальным значением, например так:

```
Private Sub Form_Resize()
  If Me.Width > 2800 Then Me.Width = 2800
End Sub
```

Однако такой способ все же дает возможность изменить размеры формы за пределы установленных значений — он просто возвращает форму обратно в ее границы, когда вы отпускаете клавишу мыши. Кроме того, если щелкнуть кнопку "развернуть", то форма

раскроется, займет все пространство экрана, а потом появится сообщение об ошибке — нельзя изменять размеры формы в режиме "развернуть". Конечно же, VB предлагает несколько путей решения данной проблемы программным образом, но они не всегда достаточно корректны. Так, форма может возникать в различных частях экрана по мере выполнения кода, который изменяет ее размеры. В Руководствах по графическому интерфейсу говорится, что пользователь всегда прав, поэтому лучше сделать так, чтобы он не мог никаким образом перейти за указанные границы формы.

Для этой цели существует бесплатный элемент управления ActiveX, называемый ARFormExtender. Поместите этот невидимый компонент на форму и установите его свойства MaxHeight, MinHeight, MaxWidth, MinWidth и это все! Теперь, запустив программу на выполнение, вы не сможете даже перетащить границы формы за пределы заданных значений. Форма останется неизменной, если вы также щелкните кнопку "развернуть".

Кроме того, ARFormExtender обрабатывает свойство ResizeContents. Если установить последнее как True, элемент управления ARFormExtender будет изменять размеры и взаимное расположение всех компонентов, расположенных на форме, в соответствии с изменением размеров самой формы. Кроме того, любое изменение размеров формы запускает собственное событие Resize элемента управления ARFormExtender, которое передает параметры WidthChange и HeightChange. Эти две величины отражают вертикальное и горизонтальное изменения (в твипах), произошедшие со времени последнего запуска события Resize.

Загрузить элемент управления ARFormExtender или получить более подробную информацию о его возможностях можно по адресу: http://www.sevillaonline.com/ActiveX/.

Совет 221. Как показать стандартное диалоговое окно Properties для файла

Если вы хотите создать стандартное Windows-приложение, желательно, чтобы оно имело диалоговое окно Properties, которое обычно вызывается командой File|Properties. Для этого можно воспользоваться API-функцией ShellExecuteEx:

```
Option Explicit

Private Type SHELLEXECUTEINFO
  cbSize As Long
  fMask As Long
  hWnd As Long
  lpVerb As String
  lpFile As String
  lpParameters As String
  lpDirectory As String
  nShow As Long
  hInstApp As Long
  lpIDList As Long
  lpClass As String
  hkeyClass As String
  dwHotKey As Long
  hIcon As Long
  hprocess As Long
End Type

Private Declare Function ShellExecuteEx Lib _
  "shell32" (lpSEI As SHELLEXECUTEINFO) As Long
Private Const SEE_MASK_INVOKEIDLIST = &HC
```

```
Private Sub Form_Click()
  Call ShowFileProperties("c:\windows\system\msvbvm50.dll")
End Sub

Private Sub ShowFileProperties(ByVal aFile As String)
  Dim sei As SHELLEXECUTEINFO
  sei.hWnd = Me.hWnd
  sei.lpVerb = "properties"
  sei.lpFile = aFile
  sei.fMask = SEE_MASK_INVOKEIDLIST
  sei.cbSize = Len(sei)
  ShellExecuteEx sei
End Sub
```

Совет 222. Настройка цветов и шрифтов в строке состояния приложения

Для настройки шрифтов и цветов в создаваемой вами строке состояния можно воспользоваться элементом управления PictureBox и API-функцией SendMessage. Каждая строка состояния может отображать объект Picture, поэтому сначала вы устанавливаете необходимые элементы Background, Foreground и Font для невидимого компонента PictureBox, а затем присваиваете изображение PictureBox объекту Picture в строке состояния. Объект Panel работает со свойством Width вместо свойства Height (Высота), в то время как функция SendMessage может получать значение этой высоты.

Поместите элемент управления PictureBox на форму и установите его свойства следующим образом:

```
Свойство       Значение
Name           picStatus
AutoRedraw     True
Visible        False
```

Перед вызовом подпрограммы PanelText измените объект Font строки состояния в соответствии с параметрами объекта Panel:

```
Option Explicit

Private Type RECT
  Left As Long
  Top As Long
  Right As Long
  Bottom As Long
End Type

Private Declare Function SendMessage Lib _
  "user32" Alias "SendMessageA" (ByVal hWnd _
  As Long, ByVal wMsg As Long, ByVal wParam _
  As Long, lParam As Any) As Long
Private Const WM_USER = &H400
Private Const SB_GETRECT = (WM_USER + 10)

Private Sub PanelText(sb As StatusBar, Index As _
  Long, aText As String, bkColor As Long, fgColor As Long)
  Dim R As RECT
  SendMessage sb.hWnd, SB_GETRECT, Index - 1, R
  With picStatus
    Set .Font = sb.Font
    .Move 0, 0, (R.Right - R.Left + 1) * _
```

```
      Screen.TwipsPerPixelX, (R.Bottom - _
      R.Top + 1) * Screen.TwipsPerPixelY
    .BackColor = bkColor
    .Cls
    .ForeColor = fgColor
    picStatus.Print aText
    sb.Panels(Index).Text = aText
    sb.Panels(Index).Picture = .Image
  End With
End Sub

Private Sub Form_Load()
   PanelText StatusBar1, 1, "Сообщение 1", _
     QBColor(1), QBColor(11)
End Sub
```

Совет 223. Как выровнять текст в командной кнопке

Вам никогда не приходилось выравнивать текст в командной кнопке? Если да, то вы, наверняка, добавляли соответствующие пробелы в свойстве Caption. Однако это можно сделать и по-другому с помощью API-функции SetWindowLong. Введите следующий код в стандартный BAS-модуль, а затем вызовите его, передавая в качестве аргументов командную кнопку и необходимую комбинацию значений для вертикального и горизонтального выравнивания:

```
Option Explicit

Private Declare Function GetWindowLong Lib _
  "user32" Alias "GetWindowLongA" (ByVal hWnd _
  As Long, ByVal nIndex As Long) As Long
Private Declare Function SetWindowLong Lib _
  "user32" Alias "SetWindowLongA" (ByVal hWnd _
  As Long, ByVal nIndex As Long, ByVal dwNewLong _
  As Long) As Long

Private Const BS_LEFT As Long = &H100
Private Const BS_RIGHT As Long = &H200
Private Const BS_CENTER As Long = &H300
Private Const BS_TOP As Long = &H400
Private Const BS_BOTTOM As Long = &H800
Private Const BS_VCENTER As Long = &HC00

Private Const BS_ALLSTYLES = BS_LEFT Or _
  BS_RIGHT Or BS_CENTER Or BS_TOP Or _
  BS_BOTTOM Or BS_VCENTER

Private Const GWL_STYLE& = (-16)

Public Enum bsHorizontalAlignments
  bsLeft = BS_LEFT
  bsRight = BS_RIGHT
  bsCenter = BS_CENTER
End Enum

Public Enum bsVerticalAlignments
  bsTop = BS_TOP
  bsBottom = BS_BOTTOM
  bsVCenter = BS_VCENTER
End Enum

Public Sub AlignButtonText(cmd As CommandButton, _
```

```
  Optional ByVal HStyle As bsHorizontalAlignments = _
  bsCenter, Optional ByVal VStyle As _
  bsVerticalAlignments = bsVCenter)

  Dim oldStyle As Long
  ' получает текущий стиль
  oldStyle = GetWindowLong(cmd.hWnd, GWL_STYLE)
  ' очищает существующие установки выравнивания
  oldStyle = oldStyle And (Not BS_ALLSTYLES)
  ' устанавливает новый стиль и
  ' обновляет кнопку
  Call SetWindowLong(cmd.hWnd, GWL_STYLE, _
    oldStyle Or HStyle Or VStyle)
  cmd.Refresh
End Sub

Private Sub Form_Load()
  Call AlignButtonText(Command1, bsLeft, bsTop)
End Sub
```

Совет 224. Управление шириной столбцов элемента управления DBGrid в процессе выполнения программы

DBGrid — один из самых популярных элементов управления для вывода данных в виде таблицы. Однако если изменить ширину столбцов сетки в процессе выполнения программы, Visual Basic не запомнит текущие значения и при следующем запуске программы выведет таблицу в неизменном виде. Здесь мы приводим текст подпрограммы, с помощью которой можно сохранять ширину столбцов для повторных запусков:

```
Option Explicit

Sub DBGridLayout(Operation As String)
  'сохраняет ширину столбцов
  Dim lWidth As Long
  Dim clm As Column
  Dim lDefWidth As Long
  '
  lDefWidth = DBGrid1.DefColWidth
  For Each clm In DBGrid1.Columns
    With cl
      Select Case LCase(Operation)
        Case "save"
          lWidth = .Width
          SaveSetting App.Title, "Cols" _
                  CStr(.ColIndex), lWidth
        Case "load"
          lWidth = GetSetting(App.Title, _
                  "Cols", CStr(.ColIndex), _
          lDefWidth)
          .Width = lWidth
      End Select
    End With
  Next clm
End Sub
```

Как видно из вышеприведенного текста, подпрограмма использует функции SaveSetting и GetSetting для хранения текущих значений ширины столбцов в разделе VB Регистра. Для применения данной подпрограммы в своем приложении вызовите ее из событий Load и Unload родительской формы. Затем укажите операцию, которую должна выполнить подпрограмма, как это показано ниже:

```
Private Sub Form_Load()
  DBGridLayout "Load"
End Sub

Private Sub Form_Unload(Cancel As Integer)
  DBGridLayout "Save"
End Sub
```

Совет 225. Простой способ запомнить события запуска VB-формы

Если вам трудно запомнить последовательность событий при запуске VB-формы, попробуйте запомнить следующую простую фразу:

```
I Like RAP (я люблю рэп)
```

которую можно расшифровать следующим образом:

```
I — Initialize
Like — Load
R — Resize
A — Activate
P — Paint
```

Если вам неудобно признаться в том, что вы любите рэп, можно всегда заменить эту фразу на другую, например: I Loathe RAP (я ненавижу рэп).

Совет 226. Обмен переменными между разными приложениями

Иногда возникает необходимость обмениваться переменными между различными приложениями, например при совместном использовании базы данных. Для этого создадим ActiveX EXE с модулем класса Class1. Здесь нельзя создать библиотеку ActiveX DLL, поскольку она выполняется внутрипроцессно in-process в каждом клиенте. Добавьте к проекту модель (команда Project|Add Module) и опишите переменную, которая будет передаваться в другое приложение, в качестве общей.

Код в ActiveX EXE будет выглядеть следующим образом:

```
********* Код модуля **********

Global gCon As Database
Global glngNrOfObjects As Long

********* Код класса **********

Public Property Set DBConn(Con As Database)
  Set gCon = Con
End Property

Public Property Get DBConn() As Database
  Set DBConn = gCon
End Property

Private Sub Class_Initialize()
  ' эта переменная хранит количество
  ' созданных объектов
  glngNrOfObjects = glngNrOfObjects + 1
End Sub
```

```
Private Sub Class_Terminate()
  ' уменьшение количества объектов
  glngNrOfObjects = glngNrOfObjects - 1
  ' если нет никаких объектов,
  ' связь прерывается
  If glngNrOfObjects = 0 Then
    If Not gCon Is Nothing Then
      gCon.Close
      Set gCon = Nothing
    End If
  End If
End Sub
```

Теперь установите ссылку к библиотеке ActiveX EXE в своем проекте и используйте ее, например, так:

*** Код приложения, использующего ActiveX EXE ***

```
Dim clsDBConn As Class1

Private Sub Form_Load()
  Set clsDBConn = New Class1
  ' связь устанавливается, когда
  ' создан первый объект
  If clsDBConn.DBConn Is Nothing Then
    Set clsDBConn.DBConn = Workspaces(0). _
    OpenDatabase("\VisStudio\Vb98\Biblio.mdb")
  End If
End Sub

Private Sub Form_Unload(Cancel As Integer)
  Set clsDBConn = Nothing
End Sub
```

Вам нет необходимости писать код для подсчета экземпляров объектов и для закрытия базы данных, если вы работаете с простой переменной, а не ссылкой на объект. В этом случае также поменяйте Property Set на Property Let.

И наконец, главное. Не забудьте установить ссылку к объектам DAO, так как они используют объект базы данных.

Андрей Колесов, Ольга Павлова

© 2000, Андрей Колесов, Ольга Павлова
Авторский вариант. Статья была опубликована с незначительной литературной правкой в журнале "КомпьютерПресс" N 01/2000, компакт-диск.

Совет 227. Представление числового значения прописью

Задача автоматического представления числового значения прописью является одной из наиболее часто встречающихся при составлении юридических документов. При этом речь может идти не только о денежных суммах, но и о количестве каких-либо товаров.

Кстати, многие из первых макросов, присланных для конкурса MS Office Extensions (www.microsoft.ru/offext/) еще в 1997 году, решали именно эту задачу и некоторые из них

получили призы. Но предлагаемые там решения имеют уже "встроенный" характер, к тому же, на наш взгляд, не очень универсальны и оптимальны. Исходный код у многих таких расширений закрыт, а у тех, где открыт, требует, честно говоря, слишком больших усилий, чтобы в них разобраться.

Поэтому мы решили предложить читателям собственный вариант.

В листинге 227[22] находится процедура SummaString, которая в свою очередь использует внутреннюю служебную подпрограмму SummaStringThree. Там же имеется описание входных и выходных параметров. Эта процедура учитывает правильное склонение числительных в зависимости от рода единицы измерения (мужской, женский, средний) и ее практическое применение может выглядеть следующим образом:

```
Source& = 2000011
CALL SummaString(Summa$, Source&, 1, "рубль", "рубля", "рублей")
Print Summa$ ' два миллиона одиннадцать рублей
'
Source& = 22
CALL SummaString(Summa$, Source&, 2, "копейка", "копейки", "копеек")
Print Summa$ ' двадцать две копейки
'
Source& = 1231
Call SummaString(Summa$, Source&, 3, "колесо", "колеса", "колес")
Print Summa$ ' одна тысяча двести тридцать одно колесо
'
Call SummaString(Summa$, Source&, 1, "", "", "")
Print Summa$ ' одна тысяча двести тридцать один
Print Summa$ + " руб." ' одна тысяча двести тридцать один руб.
```

Еще несколько дополнительных замечаний:

1. Для хранения исходного числа используется переменная типа Long, что позволяется работать с числами до 2147483647 (2^31-1). Если нужно увеличить диапазон, то следует просто заменить тип переменных Source и TempValue на Double или Currency, а также добавить в SummaString код для учета триллионов и т.д. (см. листинг).
2. Обратите внимание, что процедуры SummaString и SummaStringThree не используют каких-либо экзотических программных конструкций - они могут работать с любыми версиями VB, VBA и Basic/DOS.
3. По правилам оформления бухгалтерских документов сумма прописью должна обязательно начинаться с прописной буквы. Для такого преобразования можно использовать функцию StrConv, но при этом нужно помнить, что ее корректная работа с национальными алфавитами в VB до версии 5.0 включительно обеспечивается только при установке в системе соответствующей кодовой таблицы (подробнее об этом см. "Особенности работы со строками в VB"[23]).

   Чтобы выполнить нужное преобразование независимо от версии VB и системных установок Windows, гораздо проще применить следующую конструкцию:

   ```
   Mid$(Summa$, 1) = Chr$(Asc(Summa$) - 32)
   ```

[22] http://www.visual.2000.ru/develop/ms-vb/tips/0001-1.htm
[23] http://www.visual.2000.ru/develop/ms-vb/cp9910/strvb-11.htm

Для DOS-овских версий, в которых преобразование регистра выполняется только для английских букв, такая операция будет выглядеть немного сложнее (альтернативная русская таблица DOS - cp866):

```
s% = ASC(Summa$)
IF s% <= 175 THEN s% = s% - 32 ELSE s% = s% = 80
MID$(Summa$, 1) = CHR$(s%)
```

Здесь мы воспользовались тем обстоятельством, что полученная символьная строка гарантированно (так работает наш алгоритм) начинается со строчной русской буквы. В противном случае пришлось бы сделать небольшую проверку на попадание ASCII- кода в нужный диапазон.

Листинг 277. Функция SummaString выполняет задачу "запись числи прописью"

```
Sub SummaString(Summa$, Source As Long, Rod%, w1$, w2to4$, w5to10$)
'
' "Сумма прописью":
'  преобразование числа из цифрого вида в символьное
' ================================================
' Исходные данные:
'  Source - число от 0 до 2147483647 (2^31-1)
   ' Если нужно оперировать с числами > 2 147 483 647
   ' замените описание переменных Source и TempValue на "AS DOUBLE"
'
'    далее нужно задать информацию о единице изменения
'  Rod%   = 1 - мужской, = 2 - женский, = 3 - средний
'     название единицы изменения:
'  w1$     - именительный падеж единственное число (= 1)
'  w2to4$  - родительный падеж единственное число (= 2-4)
'  w5to10$ - родительный падеж множественное число ( = 5-10)
'
'  Rod% должен быть задано обязательно, название единицы может быть
'       не задано = ""
' _________________-
' Результат: Summa$ - запись прописью
'
'================================
   Dim TempValue As Long
   '
   If Source& = 0 Then
     Summa$ = RTrim$("ноль " + w5to10$): Exit Sub
   End If
   '
   TempValue = Source: Summa$ = ""
   ' единицы
   Call SummaStringThree(Summa$, TempValue, Rod%, w1$, w2to4$, w5to10$)
   If TempValue = 0 Then Exit Sub
   ' тысячи
   Call SummaStringThree(Summa$, TempValue, 2, "тысяча", "тысячи", "тысяч")
   If TempValue = 0 Then Exit Sub
   ' миллионы
   Call SummaStringThree(Summa$, TempValue, 1, "миллион",
"миллиона", "миллионов")
   If TempValue = 0 Then Exit Sub
   ' миллиардов
   Call SummaStringThree(Summa$, TempValue, 1, "миллиард",
"миллиарда", "миллиардов")
   If TempValue = 0 Then Exit Sub
```

```
   '
   ' Если нужно оперировать с числами > 2 147 483 647
   ' измените тип переменных (см. выше) и добавьте эту строку для триллионов:
   ' CALL SummaStringThree(Summa$, TempValue#, 1, "трилллион",
"триллиона", "триллионов")
   ' IF TempValue# = 0 THEN EXIT SUB
   '
   ' Что идет после триллионов, я плохо представляю...
   '
End Sub

Sub SummaStringThree(Summa$, TempValue As Long, Rod%, w1$, w2to4$, w5to10$)
'
'  Формирования строки для трехзначного числа:
'  (последний трех знаков TempValue
   ' Если нужно оперировать с числами > 2 147 483 647
   ' замените в описании на  TempValue AS DOUBLE
'====================================
   Dim Rest%, Rest1%, EndWord$, s1$, s10$, s100$
   '
   Rest% = TempValue& Mod 1000
   TempValue& = TempValue& \ 1000
   If Rest% = 0 Then    ' последние три знака нулевые
     If Summa$ = "" Then Summa$ = w5to10$ + " "
     Exit Sub
   End If
   '
   ' начинаем подсчет с Rest
   EndWord$ = w5to10$
   ' сотни
   Select Case Rest% \ 100
     Case 0: s100$ = ""
     Case 1: s100$ = "сто "
     Case 2: s100$ = "двести "
     Case 3: s100$ = "триста "
     Case 4: s100$ = "четыреста "
     Case 5: s100$ = "пятьсот "
     Case 6: s100$ = "шестьсот "
     Case 7: s100$ = "семьсот "
     Case 8: s100$ = "восемьсот "
     Case 9: s100$ = "девятьсот "
   End Select
   '
   ' десятки
   Rest% = Rest% Mod 100: Rest1% = Rest% \ 10
   s1$ = ""
   Select Case Rest1%
     Case 0: s10$ = ""
     Case 1  ' особый случай
       Select Case Rest%
         Case 10: s10$ = "десять "
         Case 11: s10$ = "одиннадцать "
         Case 12: s10$ = "двенадцать "
         Case 13: s10$ = "тринадцать "
         Case 14: s10$ = "четырнадцать "
         Case 15: s10$ = "пятнадцать "
         Case 16: s10$ = "шестнадцать "
         Case 17: s10$ = "семнадцать "
         Case 18: s10$ = "восемнадцать "
         Case 19: s10$ = "девятнадцать "
       End Select
     Case 2: s10$ = "двадцать "
     Case 3: s10$ = "тридцать "
```

```
      Case 4: s10$ = "сорок "
      Case 5: s10$ = "пятьдесят "
      Case 6: s10$ = "шестьдесят "
      Case 7: s10$ = "семьдесят "
      Case 8: s10$ = "восемьдесят "
      Case 9: s10$ = "девяносто "
    End Select
    '
    If Rest1% <> 1 Then  ' единицы
      Select Case Rest% Mod 10
        Case 1
          Select Case Rod%
            Case 1: s1$ = "один "
            Case 2: s1$ = "одна "
            Case 3: s1$ = "одно "
          End Select
          EndWord$ = w1$
        Case 2
          If Rod% = 2 Then s1$ = "две " Else s1$ = "два "
          EndWord$ = w2to4$
        Case 3: s1$ = "три ": EndWord$ = w2to4$
        Case 4: s1$ = "четыре ": EndWord$ = w2to4$
        Case 5: s1$ = "пять "
        Case 6: s1$ = "шесть "
        Case 7: s1$ = "семь "
        Case 8: s1$ = "восемь "
        Case 9: s1$ = "девять "
      End Select
    End If
    '
    ' сборка строки
    Summa$ = RTrim$(RTrim$(s100$ + s10$ + s1$ + EndWord$) + " " + Summa$)
End Sub
```

Совет 228. Как практически воспользоваться функцией SummaString?

(Продолжение Совета 227[24])

Вопрос можно сформулировать по-другому: как получить исходное числовое значение Source, которое может быть задано в виде символьной цифровой строки? Для этого можно также использовать функцию ResultSumma$ (листинг 228[25]), которая сначала выполняет необходимые проверки на допустимость преобразования символьной строки в число, а уже потом обращается к SummaSrting.

Например, вы хотите реализовать при работе с документами Word 97/2000 такую дополнительную функцию: выделив некое число, записанное цифрами, легким движением приписать в скобках после этого числа его представление прописью.

Для этого в среде VBA (в нашем случае данный пример реализован в виде документа Summa.doc) необходимо загрузить BAS-модуль с уже созданными нами процедурами ResultString, SummaString и SummaStringThree, а потом написать такую макрокоманду:

```
Sub ЧислоПрописью()
  '
  Dim Summa$
  Summa$ = ResultSumma$(Selection.Text, 1, "", "", "", 0)
```

[24] http://www.visual.2000.ru/develop/ms-vb/tips/0001-1.htm
[25] http://www.visual.2000.ru/develop/ms-vb/tips/0001-1.htm

```
  If Summa$ <> "" Then ' допустимое значение
    Selection.Text = Selection.Text + " (" + Summa$ + ") "
  End If
End Sub
```

Тогда, выделив в документе, например строку "1732", и обратившись к макрокоманде, вы увидите в тексте: "1732 (одна тысяча семьсот тридцать два)".

При желании можно также написать автономную утилиту (Summa.vbp), которая выполняет нужное преобразование на основе данных, введенных в диалоговом режиме (рис. 228). Обратите внимание, эта утилита сохраняет все параметры, введенные при предыдущем запуске. Данный VB-проект несложно преобразовать в отдельный ActiveX-сервер или интегрировать в среду приложения, поддерживающего технологию VBA.

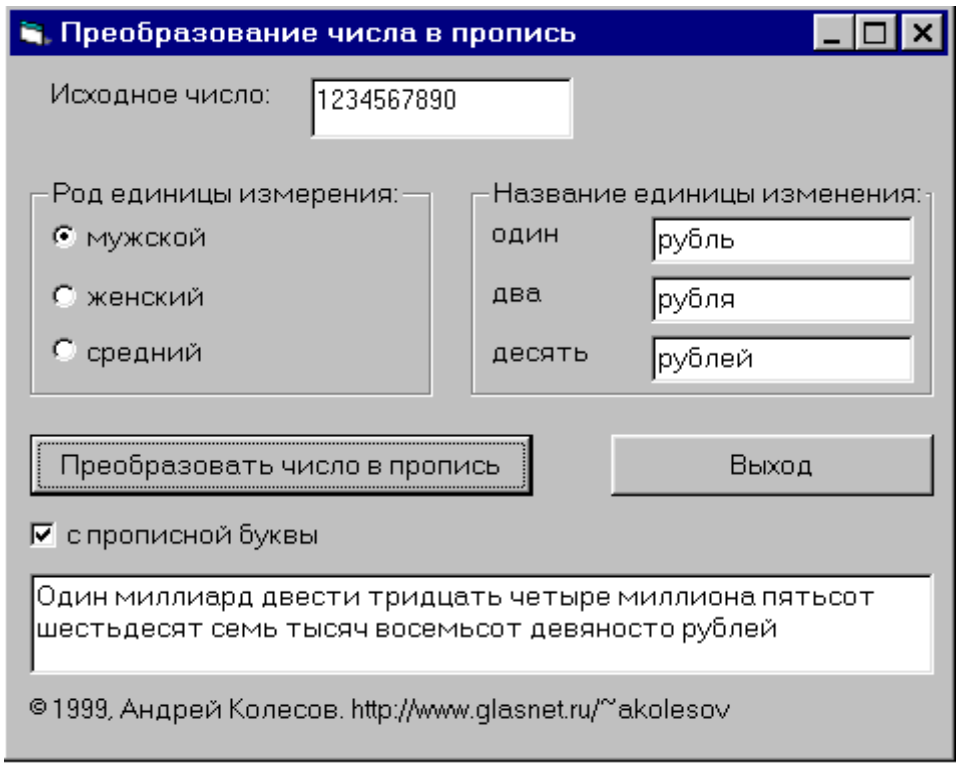


Рис. 228

Короче говоря, у разработчика открываются широкие возможности по созданию собственного варианта встроенного использования приведенных здесь процедур. Нам кажется, что дальше читатели смогут уже самостоятельно реализовать и такой вариант процедуры или макроса, который, например, преобразует выражение типа "100 руб. 21 коп." в формат "Сто руб. 21 коп.". Будут проблемы - пишите.

Листинг 2. Вспомогательный вариант "сумма прописью"

```
Public Function ResultSumma$(Source$, Rod%, w1$, w2to4$, w5to10$, iCase%)
  Dim i%, Summa$
  ' Проверка правильности числа
  ' и преобразование его в пропись
  '
  ' Source$ - цифровая запись числа в символьном виде
  ' Rod, w1, w2to4, w5to10  - см. SummaString
  ' iCase > 0  - первую букву преобразовать в прописную
```

```
  ' ========================

  If Source$ = "" Then
    MsgBox "Пустая символьная строка"
    Exit Function
  End If
  For i = 1 To Len(Source$)
    If Not Mid$(Source$, i, 1) Like "[0-9]" Then
      MsgBox "Исходная строка содержит не цифры:" & vbCrLf & Source$
      Exit Function
    End If
  Next
  If Val(Source$) > &H7FFFFFFF Then
      MsgBox "Превышен предел - 2147483647"
      Exit Function
  End If
  Call SummaString(Summa$, CLng(Val(Source$)), Rod%, w1$, w2to4$, w5to10$)
  If iCase% > 0 Then  ' написать с прописной буквы
    Mid$(Summa$, 1) = Chr$(Asc(Summa$) - 32)
  End If
  ResultSumma$ = Summa$
End Function
```

Андрей Колесов, Ольга Павлова

© 2000, Андрей Колесов, Ольга Павлова
Авторский вариант. Статья была опубликована с незначительной литературной правкой в журнале "КомпьютерПресс" N 01/2000, компакт-диск.

Совет 229. Используйте реентерабельные и рекурсивные процедуры. Но осторожно!

Реентерабельные процедуры (reentrable, с возможностью повторного входа) — это процедуры, которые позволяют использовать одну копию программы для обработки нескольких одновременных запросов. Если речь идет об однопроцессорной системе, то такая ситуация возможна в случае параллельной (точнее, псевдо-параллельной) работы отдельных исполняемых модулей, которые обращаются в одной библиотеке: первая программа обращается к процедуре некой библиотеки, ее прерывает другая, более приоритетная программа и обращается к той же процедуре.

Такая же ситуация может возникнуть и в однозадачной системе, допускающей распараллеливание вычислительных процессов (для многопроцессорных компьютеров).

Подобный случай возможен и внутри одной программы, работающей на обычном ПК. Например, вы написали процедуру Integral для вычисления определенного интеграла для функции, которая задается в виде указателя (для VB какой режим стал в принципе возможен с появлением в VB 5.0 функции AddressOf). Если же вам нужно взять двойной интеграл, то может легко получиться, что подынтегральная функция в свою очередь еще раз обратится к Intergal.

Идея реентерабельности процедур основывается на реализации двух постулатов:

1. Программный код процедур не должен модифицироваться во время выполнения. В счастью, в языках высокого уровня (в том числе и в VB) такое в принципе невозможно, а

вот на ассемблере такие штуки делаются легко. Более того, в старые добрые времена подобная коррекция кода порой предлагалась даже в качестве стандартных приемом "изысканного" программирования.

2. Для каждого вызова в процедуре формируется отдельный уникальный набор переменных. Это реализуется двумя способами. В первом, переменные хранятся в буфере, который передается из вызывающей программы. Во втором, процедура сама предоставляет такой отдельный буфер, используя принцип стековой памяти.

Очевидно, что второй вариант гораздо проще в реализации, и понятно, что для VB-процедур это означает, что все внутренние переменные должны быть динамическими.

Частным, но наиболее распространенным вариантом реентерабельных процедур для однозадачных программ являются рекурсивные процедуры — те, которые в явном виде обращаются сами к себе. Использование рекурсивных алгоритмов может быть очень полезным, но следует помнить, что в общем случае может падать скорость вычислений (идет очень активное динамическое перераспределение памяти). Кроме того, глубокий уровень вложенности рекурсии может быть причиной выхода программы за пределы стекового пространства, то есть появления ошибки или даже аварийного останова системы.

Приведем несколько примеров использования рекурсивных процедур.

Пример 1. Процедура реверсивного преобразования строковой переменной (перестановка символов задом наперед). Такая встроенная функция появилась в VB 6.0 (SrtReverse) и разные способы ее реализации традиционными средствами Basic мы уже приводили (см. "Особенности работы со строковыми переменными в VB", часть 2[26]). Еще один вариант с помощью рекурсивной функции выглядит следующим образом:

```
Public Function ReverseRecursion$(s$)
  '
  ' инверсия строковой переменной с помощью
  ' рекурсивной функции
  Dim c$
  c$ = Left$(s$, 1)
  If c$ = "" Then
    ' исходная строка пустая, завершаем рекурсию
    ReverseRecursion$ = ""
  Else
    ' не пустая строка - продолжаем рекурсию
    ReverseRecursion$ = ReverseRecursion$(Mid$(s$, 2)) & c$
  End If
End Function
```

Данный вариант выглядит не сложнее, чем традиционный алгоритм:

```
Function ReverseString$ (Source$)
  Dim sReverse$, i%
  For i = Len(Source$) To 1 Step -1
          sReverse = sReverse & Mid$(Source, i, 1)
  Next I
  ReverseString$ = sReverse
End Function
```

[26] http://www.visual.2000.ru/develop/ms-vb/cp9910/strvb-2.htm

Однако при этом рекурсивный алгоритм работает примерно в 3,5-4 раза медленнее. Обратите внимание, что все внутренние переменные ReverseRecursion (в данном случае это одна переменная c$) являются динамическими. Если же поставить ключевое слово Static в описании функции, то она будет просто неверно работать. (Попробуете сами — будет выдаваться пустая строка! Это объясняется просто — рекурсия заканчивается, когда переменная c$="", и именно она участвует в "обратном" формировании строки.)

Пример 2. Вычисление факториала N! = N * (N-1) *... 1 Это является классическим примером рекурсивного соотношения, которое записывается следующим образом:

```
N! = N * (N-1)!
0!=1
```

Такое вычисление также может быть реализовано в виде двух алгоритмов:

1) рекурсивный:

```
Public Function FactorialRecursion&(N%)
  ' вычисление факториала методом рекурсии
  If N% = 0 Then ' Завершаем рекурсию - 0!
    FactorialRecursion& = 1 ' 0! = 1
  Else ' продолжаем рекурсию
    FactorialRecursion& = N% * FactorialRecursion&(N% - 1)
  End If
End Function
```

2) обычный цикл:

```
Public Function Factorial&(N%)
  ' вычисление факториала методом рекурсии
  Dim Fact&, i%
  Fact& = 1
  If N% > 1 Then ' нетривиальный вариант
   For i% = 2 To N%
     Fact& = Fact& * i%
   Next
  End If
  Factorial& = Fact&
End Function
```

Код в рекурсивной функции заметно короче, но работает в 4 раза медленнее.

Если вы сравните два варианта решения данной задачи, то увидите, что в них используется, казалось бы, один алгоритм - следующее значение функции вычисляется с помощью результата, полученного на предыдущем шаге. Однако при ближайшем рассмотрении мы обнаружим принципиальную разницу: при использовании цикла "значение предыдущего шага" уже известно, а в случае рекурсии — его еще только предстоит вычислить.

Представленные выше примеры являются слишком тривиальными, так как тут речь идет о простых линейных структурах заданной длины. В этом случае использование обычных циклов гораздо эффективнее (мы привели здесь рекурсию чисто из познавательных соображений). Но есть случаи когда рекурсия просто необходима — например, при обработке древовидных структур, когда, двигаясь от корня дерева, вам нужно пройти все его ветки, число и глубина которых заранее неизвестны. Классическим примером здесь является задача передвижения по каталогам файловой системы, которую мы рассмотрим в следующем совете.

Совет 230. Используйте рекурсию для передвижения по файлам

Предположим, мы хотим определить число всех файлов, соответствующих заданному шаблону имени, в некотором каталоге. А заодно и количество вложенных подкаталогов. Такая задача решается с помощью процедуры HowManyFilesInThisDirectory, приведенной в листинге 230[27], а обращение к ней может выглядеть следующими образом:

```
PathName$ = "d:\qb_lib\" ' стартовый каталог
FileName$ = "*.bas" ' шаблон имени файла
FileCounter% = 0 ' начальное значение счетчика найденных файлов
DirCounter% = 0  ' начальное значение счетчика просмотренных
                                                ' подкаталогов
Call HowManyFilesInThisDirectory(PathName$, FileName$, FileCounter%)
```

Алгоритм этой процедуры достаточно прост — сначала выполняется поиск имен файлов в заданном каталоге, там же определяется список подкаталогов, а потом в цикле процедура обращается сама к себе с новым именем каталога. Тут все понятно, но следует обратить внимание на следующие моменты:

1. Результат работы процедуры формируется в двух переменных: FileCounter и DirCounter. Однако первая передается в качестве параметра функции (и постоянно модифицируется при этом), а вторая является глобальной, статической переменной рекурсивной функции.
2. При просмотре содержимого каталогов, казалось бы, проще использовать такую конструкцию:

```
3.   MyDir$ = Dir(PathName$, vbDirectory)
4.   Do While MyDir$ <> ""
5.     If MyDir$ <> "." And MyDir$ <> ".." Then
6.       If GetAttr(PathName$ + MyDir$) = vbDirectory Then
7.         ' найден каталог
8.         DirCounter% = MyDirCounter% + 1
9.         NewPathName$ = PathName$ + MyDir$(i) + "\"
10.        Call HowManyFilesInThisDirectory(NewPathName$, _
11.          FileName$, FileCounter%)
12.      End If
13.    End If
14.    MyDir$ = Dir ' следующий поиск
15.  Loop
```

Однако такой код будет неправильным. Дело в том, что функция Dir не является реентерабельной. Более того, ее применение основано на цикле последовательных обращений, когда первое обращение (вызов функции с параметрами) задает шаблон поиска (который запоминается внутри функции в ее статической переменной), а затем (вызов функции без параметров) происходит как бы продолжение первоначально заданной операции.

Приведенный выше код нарушает такой порядок обращения — еще до завершения цикла по текущему шаблону управление передается в процедуру HowManyFilesInThisDirectory, где инициализируется новая операция первого поиска. Поэтому необходимо сначала составить полный список подкаталогов (процедура CurrentDirCounter), а уже потом, перебирая его в цикле, рекурсивно обращаться на обработку очередного каталога.

[27] http://www.visual.2000.ru/develop/ms-vb/tips/0001-2.htm

16. В процедуре формирования списка элементов каталога в массиве мы используем конструкцию:

```
17.  If MyDirCount% > UBound(arrPath$) Then
18.    ReDim Preserve arrPath$(UBound(arrPath$) + 100)
19.  End If
```

Это обеспечивает автоматическое увеличение начального размера динамического массива с сохранением ранее записанной информации.

20. Здесь с сожалением можно отметить, что в VB не допускается передача адресов функций в качестве параметров VB-процедуры (AddressOf можно использовать только при обращении к DLL). Если бы это было возможно, то мы могли бы использовать одну и ту же процедуру HowManyFilesInThisDirectory для выполнения разнообразных операций с найденными файлами (например, контекстного поиска в них). А пока нам придется брать исходный код этой процедуры и "вшивать" нужные операции, создавая новые варианты программ.

Листинг 230. Перемещение по дереву файловой системы

```
Public DirCounter% ' текущее число просмотренных каталогов

Public Sub HowManyFilesInThisDirectory(PathName$, FileName$, FileCounter%)
'
'  подсчет числа найден файлов по заданному шаблону
'  PathName$ - каталог
'  FileName$ - шаблон имени файла
'  FileCounter% - текущее значение счетчика найденых файлов
'================================================
   Dim MyFile$, MyDirCount%
   Dim NewPathName$, i%

   DirCounter% = DirCounter% + 1 ' смотрим очередной каталог
   '
   ' подсчет файлов в данном каталоге:
    MyFile$ = Dir(PathName$ + FileName$) ' первый поиск
    Do While MyFile$ <> ""
      FileCounter% = FileCounter% + 1
      ' тут можно выполнить какую-нибуль операцию с файлом
      MyFile$ = Dir  ' следующий поиск
    Loop
    '
    ' определяем состав подкаталогов в данном каталоге
    ReDim arrPath$(100)  ' для списка подкаталов
    Call CurrentDirCounter(PathName$, "", MyDirCount%, arrPath$(),
vbDirectory)
    '
    If MyDirCount% > 0 Then 'есть подкаталоги
      For i% = 1 To MyDirCount%
        ' !! рекурсивное обращение к САМОЙ СЕБЕ!!
        NewPathName$ = PathName$ + arrPath$(i) + "\"
        Call HowManyFilesInThisDirectory(NewPathName$, FileName$,
FileCounter%)
      Next
    End If
End Sub

Public Sub CurrentDirCounter(PathName$, FileName$, MyDirCount%,
arrPath$(), attr%)
'
'  Формирование списка имен элементов (attr% задает тип)
```

```
'  в текущем каталоге
'
    Dim MyDir$

    MyDirCount% = 0   'счетчик подкаталов в текущем каталоге
    MyDir$ = Dir(PathName$ + FileName$, attr%) 'первый поиск подкаталогов
    Do While MyDir$ <> ""
      If MyDir$ <> "." And MyDir$ <> ".." Then
        If GetAttr(PathName$ + MyDir$) = attr% Then  ' найден каталог
          MyDirCount% = MyDirCount% + 1
          If MyDirCount% > UBound(arrPath$) Then
            ' увеличивает размер массива с сохранением старой информации
            ReDim Preserve arrPath$(UBound(arrPath$) + 100)
          End If
          arrPath$(MyDirCount%) = MyDir$
         ' Debug.Print MyDir$; GetAttr(PathName$ + MyDir$)
        End If
      End If
      MyDir$ = Dir  ' следующий поиск
    Loop
    '
End Sub
```

Совет 231. Преобразование имен файлов в определенный регистр

Традиционно системы DOS/Windows являются нечувствительными к регистру символов в именах файлов и каталогов: имена FileName, FiLENAME и т.д. являются идентичными. Однако в Unix-системах это не так, и мы, например, столкнулись с этой проблемой, когда начали заниматься созданием Web-узла, переписывая файлы с нашего локального компьютера на сервер.

Мы приняли однозначное правило — имена элементов оглавления пишутся только строчными буквами. Однако, если соблюдение этого правила в тексте HTML-файла не вызывает никаких сложностей — там все видно, то управление физическими названиями в оглавлении оказалось не так просто. Так, вид имени файла в окне Windows Explorer довольно часто совершенно не соответствует тому, каким он является на самом деле (не вполне понятно, почему).

Решением этой проблемы стало создание процедуры, которая автоматически просматривает заданные каталоги и производит замену всех символов в именах файлов и подкаталогов на строчные буквы.

Работа с отдельным файлов выглядит достаточно просто:

```
' PathName$ - имя каталога
' MyFile$ - исходный файл
NewFile$ = LCase$(MyFile$)
If NewFile$ <> MyName$ Then ' нужна замена имени
  Name PathName$ + MyName$ As PathName$ + NewFile$
End If
```

А как выполнить такую операцию для всех файлов по заданному шаблону? Казалось бы, это можно сделать так:

```
MyFile$ = Dir(PathName$ + FileName$) ' первый поиск
Do While MyFile$ <> ""
  '  анализ имени и переименование (см. код выше)
```

```
  MyDir$ = Dir ' следующий поиск
Loop
```

Но на самом деле такая конструкция работать не будет, опять же из-за отсутствия реентерабельности Dir. Действительно, еще до завершения операции с заданным шаблоном мы производим коррекцию списка элементов каталога. Поэтому корректная реализация изменения имен файлов в списке выглядит следующим образом (аналогично обработке подкаталогов в примере, приведенном выше):

```
ReDim arrPath$(100) ' для списка файлов
Call CurrentDirCounter(PathName$, MyDirCount%, _
  arrPath$(), vbNormal)
'
If MyDirCount% > 0 Then ' есть подкаталоги
  For i% = 1 To MyDirCount%
    ' анализ имени и переименование (см. код выше)
  Next
End If
```

Совет 232. Используйте Preserve для сохранения содержимого массива

При работе с массивами довольно часто встречается ситуация, когда оказывается, что его размеров не хватает для реального объема данных и нужно увеличить эти размеры с сохранением уже записанной информации. Типичным примером является запись в массив текстового файла, число строк которого неизвестна (другой случай, связанный с формированием списка элементов каталога, приведен в Совете 230[28], процедура HowManyFilesInThisDirectory). В этом случае следует использовать ключевое слово Preserve в операторе Redim:

```
StartSize = 100 ' начальное значение массива
ReDim arrStrings$(1 To StartSize)
AddSize = 20 ' дискретность увеличения длины
Open FileName$ For Input As #1
iLen = 0 ' текущая позиция для записи
While Not EOF(1)
  iLen = iLen + 1
  If iLen > UBound(arrStrings$) Then
     ReDim Preserve arrStrings$(UBound(arrStrings$) + AddSize)
  End If
  Line Input #1, arrStrings$(iLen)
Wend

Close #1
'
' Если хотите, то после окончания ввода можно уменьшить
' размеры массива до реального числа введенных элементов
ReDim Preserve arrStrings$(iLen)
```

Коррекция длины массива с использование Preserve возможна и для многомерных массивов, но при этом допускается изменение только последнего индекса (что связано с порядком хранения многомерных данных — по строкам). Например:

```
ReDim SomeArray(10, 10, 10)
ReDim Preserve SomeArray(10, 10, 15) ' допустимо
ReDim Preserve SomeArray(10, 15, 10) ' недопустимо
```

[28] http://www.visual.2000.ru/develop/ms-vb/tips/0001-2.htm

Совет 233. Используйте FreeFile для получения номера файла. Но будьте при этом бдительны!

При работе с файлами можно использовать два метода их нумерации: статический, когда номер файла задается константой, например #2, или динамический, когда номер выбирается из списка свободных и хранится в переменной:

```
FileNumber = FreeFile
Open FileName$ For Input As #FileNumber
Line Input #FileNumber, a$
```

Динамический вариант обеспечивает защиту от ситуации, когда номер файла назначается повторно. Но здесь же кроется и определенная опасность: при работе с большим числом динамически открываемых файлов нужно четко следить за их своевременным закрытием. (Номер файла закрепляется за файлом момент открытия и освобождается в момент закрытия. Поэтому обращение к FreeFile нужно выполнять непосредственно перед оператором Open.) Во избежание таких проблем желательно при отладке следить за числом открытых файлов в программе.

Но самое главное — в одной программе нельзя использовать одновременно два метода. В крайнем случае нужно в начале программы открыть все файлы со статическими номерами, а уже потом работать с динамически выделяемыми адресами.

Андрей Колесов, Ольга Павлова

© 2000, Андрей Колесов, Ольга Павлова
Авторский вариант. Статья была опубликована с незначительной литературной правкой в журнале "КомпьютерПресс" N 2/2000, CD-ROM.

Совет 234. Используйте свойство ItemData элемента управления ListBox для хранения идентификаторов

Часто при заполнении элемента управления ListBox информацией из базы данных бывает необходимо, чтобы этот компонент не только выводил описания элементов базы данных, но и хранил некоторые ключевые значения. Например, если окно списка содержит имена сотрудников, наверное, будет полезно хранить здесь также их персональные идентификаторы. В том случае, если вы используете числовые идентификаторы, VB предоставляет возможность осуществить это с помощью свойства ItemData компонента ListBox, которое позволяет хранить дополнительное число для каждого элемента списка. Проиллюстрируем это на таком примере. Поместите окно списка на стандартную форму и введите следующий код:

```
Private Sub Form_Load()
  ' Заполнение списка и массива ItemData
  ' соответствующими элементами
  With List1
    .AddItem "Владимир Николаев"
    .ItemData(List1.NewIndex) = 42310
    .AddItem "Максим Сергеев"
    .ItemData(List1.NewIndex) = 52855
    .AddItem "Михаил Смирнов"
    .ItemData(List1.NewIndex) = 64932
```

```
    .AddItem "Анна Васильева"
    .ItemData(List1.NewIndex) = 39227
  End With
End Sub

Private Sub List1_Click()
  Dim Msg As String
  ' Вывод окна сообщения, содержащего
  ' идентификатор и имя сотрудника
  With List1
    Msg = .ItemData(.ListIndex) & " " & .List(.ListIndex)
    MsgBox Msg
  End With
End Sub
```

Запустите проект на выполнение — в окне списка выводятся только имена сотрудников. Щелкните любой из элементов, и на экране появится окно сообщения, содержащее имя выбранного сотрудника и его идентификатор.

Совет 235. Как правильно сгенерировать значения ADO RecordCount в VB

Как известно, свойство ADO RecordCount возвращает количество записей в наборе записей ADO. Однако в некоторых случаях данное свойство возвращает значение, равное -1. Почему так происходит? Дело в том, что значение, возвращаемое свойством RecordCount, зависит от типа курсора в наборе записей: -1 для курсора, перемещаемого только вперед; реальное количество записей — для статического курсора или курсора, управляемого клавишами на клавиатуре; -1 или реальное количество записей — для динамического курсора в зависимости от источника данных.

Однако вам будет также интересно узнать, что значение RecordCount равно -1 для наборов записей, созданных с помощью метода Execute для объекта Connection или Command. Это происходит потому, что данный метод генерирует набор записей, перемещаться по которому можно только вперед и который возвращает, как мы уже упоминали выше, -1. В качестве примера создайте следующую подпрограмму внутри стандартного VB-проекта. (Не забудьте установить ссылку на Microsoft ActiveX Data Objects 2.1 Library и изменить путь к базе данных Biblio.mdb в соответствии с тем, где она хранится на вашем компьютере.) Запустите проект на выполнение и вы увидите окно сообщения, содержащее такие значения: -1 для набора записей на базе myConRst и 6246 — для myKeyRst.

```
Sub TestRecordCount()
  Dim myConn As ADODB.Connection
  Dim myComm As String
  Dim myConRst As ADODB.Recordset
  Dim myKeyRst As ADODB.Recordset
  Dim sConnection As String
  sConnection = "Provider=Microsoft.Jet.OLEDB.4.0;" & _
          "Data Source=D:\Microsoft Visual Studio\VB98\Biblio.mdb"

  Set myConn = New ADODB.Connection
  Set myKeyRst = New ADODB.Recordset

  myConn.Open sConnection
  myComm = "Select * From Authors"
  Set myConRst = myConn.Execute(myComm, , adCmdText)

  myKeyRst.Open myComm, myConn, adOpenKeyset
  MsgBox "RecCount from Connection: " & myConRst.RecordCount & _
          vbCr & "From Recordset: " & myKeyRst.RecordCount
```

```
  Set myKeyRst = Nothing
  Set myConRst = Nothing
  Set myConn = Nothing

End Sub
```

Совет 236. Пусть VB сам определяет, есть ли в дисководе CD-ROM компакт-диск или нет

Для того чтобы быстро определить, не забыл ли пользователь вставить компакт-диск в дисковод CD-ROM, используйте свойство IsReady для объекта Drive. Данное свойство возвращает значение True (Истина) только в том случае, если компакт-диск находится в устройстве CD-ROM. Рассмотрим следующий пример. Установите ссылку к библиотеке Microsoft Scripting Runtime (scrrun.dll), а затем создайте переменную Drive для дисковода CD-ROM. Протестируйте свойство IsReady, как показано ниже:

```
Dim FSO As FileSystemObject
Dim CDDrive As Drive

Set FSO = New FileSystemObject
Set CDDrive = FSO.GetDrive("E:")
If CDDrive.IsReady Then
  MsgBox CDDrive.VolumeName
Else
  MsgBox "Вставьте, пожалуйста, компакт-диск"
End If

Set CDDrive = Nothing
Set FSO = Nothing
```

Совет 237. Как добавить новый элемент к системному меню формы

В некоторых случаях вам может захотеться добавить дополнительный элемент к системному меню формы. (Кнопка системного меню находится в верхнем левом углу формы.) Для этого воспользуйтесь тремя API-функциями — GetSystemMenu, AppendItemMenu и DrawMenuBar:

```
Option Explicit
Private Declare Function DrawMenuBar Lib "user32" _
  (ByVal hWnd As Long) As Long

Private Declare Function GetSystemMenu Lib "user32" _
  (ByVal hWnd As Long, ByVal bRevert As Long) As Long

Private Declare Function AppendMenu Lib "user32" Alias _
  "AppendMenuA" (ByVal hMenu As Long, ByVal wFlags _
  As Long, ByVal wIDNewItem As Long, ByVal lpNewItem _
  As Any) As Long

Const MF_STRING = &H0&

Private Sub Form_Load()
  Dim SysMenu As Long
  SysMenu = GetSystemMenu(Me.hWnd, False)
  If SysMenu Then
    AppendMenu SysMenu, MF_STRING, 0, "Свой элемент меню"
    DrawMenuBar Me.hWnd
  End If
```

```
End Sub
```

Как видно из приведенного выше кода, подпрограмма находит указатель к копии системного меню текущей формы. Затем она добавляет строку "Свой элемент меню" к списку команд системного меню, и наконец, перерисовывает меню, добавляя новый текст в конец списка.

Команду DrawMenuBar следует использовать всегда при изменении системного меню, даже если в данный момент оно является невидимым.

То, что вы просто добавили новую команду к системному меню, совершенно не означает, что она будет выполнять какую-либо операцию: для этого необходимо написать соответствующий код. Функция AppendMenu предлагает несколько различных параметров, описывающих элемент, который вы хотите добавить к системному меню. Полный список этих параметров приводится в файле Справки.

Следует помнить также, для функции AppendMenu имеется более мощный аналог — новая API-функция InsertMenuItem, которая хотя и предлагает более широкие возможности управления, в то же время является намного более сложной в использовании. Поэтому если вы просто хотите поместить дополнительный элемент в конец списка команд системного меню, для этого лучше всего подойдет функция AppendMenu.

Совет 238. Как быстро осуществить проверку на наличие определенного элемента в списке

Если вы хотите определить, находится ли в списке какой-либо определенный элемент, обычно вы создаете массив, в котором и осуществляете поиск. В качестве более быстрой альтернативы для коротких списков предлагаем использовать строку неограниченной длины вместо массива. Например, предположим, что у вас есть элемент управления ListBox и вы хотите проверить, содержится ли там заданный пользователем элемент. Для начала осуществите просмотр списка в цикле и создайте такую строку:

```
For X = 0 To List1.ListCount - 1
  strItemList = strItemList & "[" & List1.List(X) & "]"
Next X
```

После этого с помощью функции Instr() определите, содержит ли новая строка искомый элемент:

```
If InStr(strItemList, "[" & strTestItem & "]") Then
  MsgBox "Уже есть в списке"
Else
  MsgBox "Добавьте в список..."
End If
```

Совет 239. Используйте для объектов ADO родные драйверы OLEDB вместо драйверов ODBC

При создании строки соединения (connection string) для объектов ADO имеется возможность задать драйвер источника данных либо как Driver, например так:

```
Driver={SQL Server};DBQ=database_file
```

либо как Provider, наподобие следующего:

```
Provider=Microsoft.Jet.OLEDB.4.0;Data Source=database_name
```

Однако в первом случае объект ADO использует более старые драйверы ODBC для связи с источником данных, в то время как во втором — применяется OLEDB, являющийся родным для ADO интерфейсом доступа к данным. Поэтому всегда, когда это возможно, пользуйтесь вариантом Provider. Напоминаем, что родные драйверы OLEDB существуют для SQL Server, Index Server, Site Server Search и Oracle.

Совет 240. Как модифицировать элемент управления DataGrid при помощи изменения набора записей

Элемент управления DataGrid, входящий в состав VB 6.0, является хорошим средством для представления данных в табличном виде. Однако в нем имеется несколько ошибок, часть из которых была исправлена в Service Patch 3, но некоторые из них все же остались. Например, если вы свяжете элементы управления DataGrid и DataEnvironment, затем измените набор данных, используемый в DataEnvironment, и обновите DataGrid с помощью метода Refresh, этот компонент останется неизменным. К сожалению, метод Refresh не работает, когда свойство DataSource элемента управления DataGrid установлено как DataEnvironment. Поэтому, чтобы отразить изменения, которые были сделаны в наборе данных элемента управления DataEnvironment, следует сначала обновить эти данные, а затем повторно связать DataGrid с DataEnvironment. Поэтому, если у вас есть кнопка Refresh, событие Click для нее может выглядеть так:

```
DataEnvironment1.rsCommand1.Requery
Set DataGrid1.DataSource = DataEnvironment1
```

Теперь, когда вы щелкните кнопку Refresh, приведенный выше код повторно свяжет элементы управления DataEnvironment и DataGrid, а затем заново заполнит последний из них обновленными данными.

Совет 241. Как запустить на выполнение VB-приложение в процессе инициализации Windows

В зависимости от версии Windows, в которой вы работаете, существуют два различных способа выполнить VB-приложение в процессе загрузки системы. Для Windows 9x поместите команду Shell в раздел [Boot] файла System.ini, например так:

```
Shell=Myprog.exe
```

Для Windows NT/2000 используйте ту же самую команду Shell в Системном Реестре (Registry) в разделе

```
HKEY_CURRENT_USER\Software\Microsoft\WindowsNT\CurrentVersion\Winlogon
```

Будьте, однако, осторожны, выполняя подобные модификации, поскольку это может привести к некорректной работе Windows.

Совет 242. Как восстановить короткое имя файла в VB без помощи API-функций

Во многих случаях может возникнуть необходимость сделать ссылку на файл в соответствии с соглашением об именах файлов 8.3. Вы, несомненно, неоднократно встречали такие имена в MS-DOS. Например, используя данное соглашение, папка Program Files превращается в Progra~1. Поэтому вам приятно будет узнать о том, что в VB 6.0 появилась возможность восстанавливать короткие имена файлов, не обращаясь к API-функции GetShortPathName. В качестве альтернативы новая библиотека Scripting Runtime предлагает свойство ShortPath для объектов File и Folder. Чтобы получить короткое имя файла, просто добавьте ссылку к Microsoft Scripting Runtime, а затем введите следующий код:

```
Private Sub Form_Load()
  Dim fsoFile As File, fso As FileSystemObject
  Set fso = New FileSystemObject
  Set fsoFile = fso.GetFile("C:\MyReallyLongName.txt")
  MsgBox fsoFile.ShortPath
  Set fsoFile = Nothing
  Set fso = Nothing
End Sub
```

Совет 243. Создавайте временные VB-файлы с помощью API-функции

Если вы когда-либо работали с Word или другим приложением Office, вы, вероятно, обратили внимание, что каждый раз при открытии файла Office создает временный файл, в котором будут храниться все изменения. И тогда у вас мог возникнуть вопрос, как создавать произвольные временные файлы в своем собственном VB-приложении. Для этого воспользуйтесь API-функцией GetTempFileName:

```
Public Declare Function GetTempFileName Lib "kernel32" _
  Alias "GetTempFileNameA" (ByVal lpszPath As String, _
  ByVal lpPrefixString As String, ByVal wUnique As Long, _
  ByVal lpTempFileName As String) As Long
```

Аргумент lpszPath передает полное имя файла (включая путь к нему), lpPrefixString позволяет добавить префикс из трех букв слева от имени файла, а wUnique дает команду Windows либо создать файл с произвольным именем (wUnique равен 0), либо использовать заданный номер. Параметр lpTempFileName, конечно же, содержит имя нового временного файла. В качестве примера поместите приведенное выше объявление API-функции в стандартный модуль, а затем напишите следующую функцию:

```
Private Function GenTempName(sPath As String)
  Dim sPrefix As String
  Dim lUnique As Long
  Dim sTempFileName As String

  If IsEmpty(sPath) Then sPath = "D:\Articles\IVB"
  sPrefix = "fVB"
  lUnique = 0

  sTempFileName = Space$(100)
  GetTempFileName sPath, sPrefix, lUnique, sTempFileName
  sTempFileName = Mid$(sTempFileName, 1, _
    InStr(sTempFileName, Chr$(0)) - 1)
  GenTempName = sTempFileName
```

```
End Function
```

После этого откройте новую форму и введите следующий код в ее событие Click. (Замените D:\Articles\IVB на любой допустимый путь.)

```
MsgBox GenTempName("D:\Articles\IVB")
```

Запустите проект на выполнение. Заданный вами каталог теперь содержит временный файл, имя которого было выведено в окне сообщения.

Обратите внимание, что для того чтобы приведенная выше функция работала надлежащим образом, вы должны задать только допустимый путь. В противном случае функция GetTempFileName вернет 0 и нулевой параметр в качестве имени файла в Windows NT. В Windows 9x неправильно указанный путь также вернет 0, а параметр lpTempFileName не будет содержать имя временного файла.

Совет 244. Как преобразовать число в строку с фиксированным количеством цифр

Порой бывает нужно вывести числа с фиксированным количеством цифр, когда спереди добавляются нули. Например, если требуется представить число 12345 в виде 0012345. Это легко осуществимо с помощью такой функции:

```
Public Function FixNumber$(SourceNumber, Lend%)
  '
  ' Преобразование числа в строку с фиксированным
  ' количеством цифр (добавляем нули спереди)
  ' ====================================
  ' SourceNumber — исходное число
  ' Lend — количество цифр
  '
  Dim MaxNumber
  MaxNumber = 10 ^ Lend
  If SourceNumber >= MaxNumber Then ' превышен предел
    ' тут могут быть разные варианты
    FixNumber$ = SourceNumber      ' вывод числа полностью
    ' или
    ' FixNumber$ = String$(Lend, "?")  ' замена на знаки вопроса
  Else ' укладывается в рамки
    FixNumber$ = Right$(MaxNumber + SourceNumber, Lend)
  End If
End Function
```

Совет 245. Используйте режим Option Explicit

Именно так назвался один из первых советов, опубликованных в самом начале цикла "Советы для тех, кто программирует на VB" четыре года назад ("КомпьютерПресс" N 5/96[29]). С тех пор мы неоднократно показывали на разных примерах пользу обязательного объявления переменных: время на написание нескольких операторов Dim многократно окупается сокращением затрат на отладку. И тем не менее, вернуться к этому вопросу вновь нас заставило письмо читателя Игоря, который обратился к нам с "неразрешимой" проблемой.

[29] http://www.visual.2000.ru/develop/ms-vb/tips/9605.htm

Игорь хотел подключить DLL-процедуру, написанную на Фортране, к VB. Вопрос корректной передачи параметров в DLL не совсем тривиален, поэтому мы посоветовали ему начать с отладки какого-нибудь простого случая, например, с передачи имени файла, который нужно прочитать в DLL, в виде строковой переменной.

С внешней стороны тестовый пример выглядел правильно:

Обращение к DLL из VB:

```
Private Declare Sub OpenFileFortran _
  Lib "FortLib" (ByVal ss As String)
  '
  Dim cc As String
  cc = "sams.txt"
  Call OpenFileFortran(cc)
End Sub
```

Процедура на Фортране:

```
Subroutine OpenFileFortran (pfn)
  character*(*) pfn
  open(0,file=pfn)
  close(0)
End
```

Прислав этот пример, Игорь сообщил, что он уже два дня безуспешно сражается с ним, причем обычно происходит аварийное завершение программы с "мгновенным исчезновением среды VB". Он испробовал разные варианты передачи имени файла, в том числе с использованием байтовых массивов и структур данных, но с тем же неудачным результатом. Распечатка переменной pfn упорно показывала отсутствие нужного имени файла. А может быть, тут проблема в путанице ANSI/Unicode? Или "глючит" Windows 98 Second Edition?

Однако "ларчик просто открывался". На самом деле в VB-программе Игоря используются две разные переменные: в первой и третьей строках указана переменная "cc" (обе буквы латинские), а во второй — "сс" (первая буква русская).

Определить это на взгляд просто невозможно, но мы всегда применяем режим Option Explicit — поэтому компилятор сразу же сообщил, что во второй строке используется неопределенная переменная. Мы, конечно, сначала не поверили своим глазам, но потом вспомнили ситуацию из времен DOS, когда у нас был установлен какой-то русификатор клавиатуры, у которого переключение клавиш на русский язык порой срабатывало только после нажатия первой клавиши. В результате при вводе такого текста

```
Andy спит
```

оказывалось, что в слове "спит" первая буква — английская. Заметить разницу было невозможно (с и с находятся на одной клавише). В этой связи еще один совет: старайтесь использовать шрифты, которые используют различные начертания внешне похожих русских и латинских букв.

После исправления идентификатора пример стал нормально работать. Таким образом, для того чтобы не терять несколько дней на отладку, и нужно-то было всего-навсего установить режим Option Explicit (который автоматически определяется командой Tools|Options|Editor|Required Variable Declaration).

Андрей Колесов, Ольга Павлова

© 2000, Андрей Колесов, Ольга Павлова
Авторский вариант. Статья была опубликована с незначительной литературной правкой в журнале "КомпьютерПресс" N 2/2000, CD-ROM.

Совет 246. Как сделать временную задержку

В версиях MS Basic для DOS имелся полезный оператор временной задержки

```
SLEEP Seconds
```

который приостанавливал выполнение программы на указанное число секунд. Такого оператора в VB не существует, но его нетрудно реализовать самостоятельно в виде следующей простой подпрограммы:

```
Sub SleepVB (Seconds)
  ' ожидание Seconds секунд
  Dim Start
  Start = Timer   ' текущее время в секундах
  Do While Timer < Start + Seconds
    ' обеспечивает параллельное выполнение других процессов
    DoEvents
  Loop
End Sub
```

Здесь нужно обратить внимание на использование функции DoEvents, которая позволяет производить другие процессы параллельно с выполнением текущего цикла. (DoEvents передает управление операционной системе, а сама функция возвращает число открытых форм в данном VB-приложении.)

Старый оператор SLEEP осуществлял задержку выполнения на заданное число секунд или до нажатия любой клавиши клавиатуры. (В Basic/DOS с ним были свои проблемы, так как код нажатой клавиши оставался в буфере клавиатуры и автоматически использовался в последующих операциях ввода. Для предотвращения подобной ситуации после оператора SLEEP нужно было очищать буфер.)

При желании такой режим можно реализовать и в нашей функции SleepVB. Один из ее вариантов, а также необходимость применения функции DoEvents мы покажем в следующем примере.

20.01.2008 напришло письмо из Самары от Юрия Гришина:

> *Я заметил, что процедура SleepVB загружает процессор на 100%, модифицировал ее следующим образом :*
>
> ```
> ' объявляем WinAPI функцию ожидания сообщения
>
> Private Declare Function WaitMessage Lib "user32.dll" () As Long
> ' объявляем WinAPI функцию ожидания сообщения
>
> Sub SleepVB(Seconds)
>   ' ожидание Seconds секунд
> ```

```
    Dim Start
    Start = Timer    ' текущее время в секундах
    Do While Timer < Start + Seconds
      ' обеспечивает параллельное выполнение других процессов
      WaitMessage ' ждем сообщения
      DoEvents ' обрабатываем
    Loop
  End Sub
```

Теперь загрузка 0 - 1% ЦП.

Совет 247. Как сделать заставку при старте VB-программы или VBA-документа

Как сделать, чтобы при запуске VB-программы появлялась некая заставка-форма, которая бы автоматически исчезала после завершения загрузки программы? Для этого лучше всего использовать вариант старта приложения с подпрограммы Main (устанавливается командой Project Properties|General|Strartup Object), которая может выглядеть примерно следующим образом:

```
Sub Main()
  '
  frmSplash.Show  ' открываем форму-заставку
  '
  Call AllStartOperations  ' этой процедурой
  ' мы обозначаем все необходимые подготовительные
  ' операции для запуска приложения, например,
  ' открываем базы данных, очищаем какие-то файлы и пр.
  '
  Load frmMain  ' загружаем основную форму приложения,
  ' с которой начнется его реальное выполнение
  Unload frmSplash  ' выгружаем форму-заставку
  'показываем основную форму и начинаем работать с ней
  frmMain.Show
End Sub
```

Обратите внимание, что форма-заставка появляется здесь только на очень короткое время (в данном случае, на время загрузки frmMain) и тут же исчезает. Как сделать временную задержку, мы покажем ниже.

А как создать заставку, которая бы появлялась при открытии офисного документа, например для Word 97/2000?

Откроем Word, создадим новый документ под названием Splash.doc и перейдем в среду VBA (Alt+F11). Затем создадим форму (Insert|UserForm), для которой установим свойства Name=frmSplash и Caption="Заставка нашей программы". На заставке можно показать разнообразную информацию о нашем приложении (название, логотип, имя автора и пр.). Мы же сейчас ограничимся только размещением небольшой GIF- картинки с помощью установки свойства Picture (рис. 247-1).

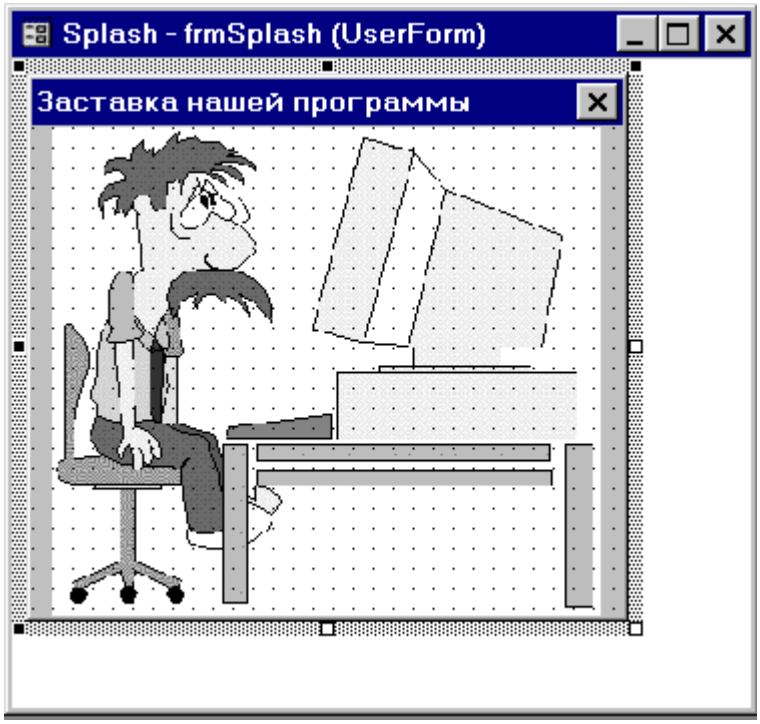


Рис. 247-1

Теперь в окне Project откроем узел Microsoft Word Objects и дважды щелкнем появившийся в нем стандартный элемент ThisDocument (в Excel это соответственно Microsoft Excel Objects|ThisWorkbook). В открывшемся окне кода в списке предопределенных событийных процедур (в правом верхнем углу) найдем подпрограмму Open. Именно ей передается управление в момент открытия документа и сюда можно поместить любой код для выполнения необходимых стартовых операций. Поместим в нее следующий программный код (рис. 247-2):

```
Private Sub Document_Open()
  ' эта процедура срабатывает при открытии документа
  '
  frmSplash.Show     ' открываем форму-заставку
  Unload frmSplash   ' выгружаем форму-заставку
  Msgbox "Hello!"    ' фиксируем начало реальной работы
End Sub
```

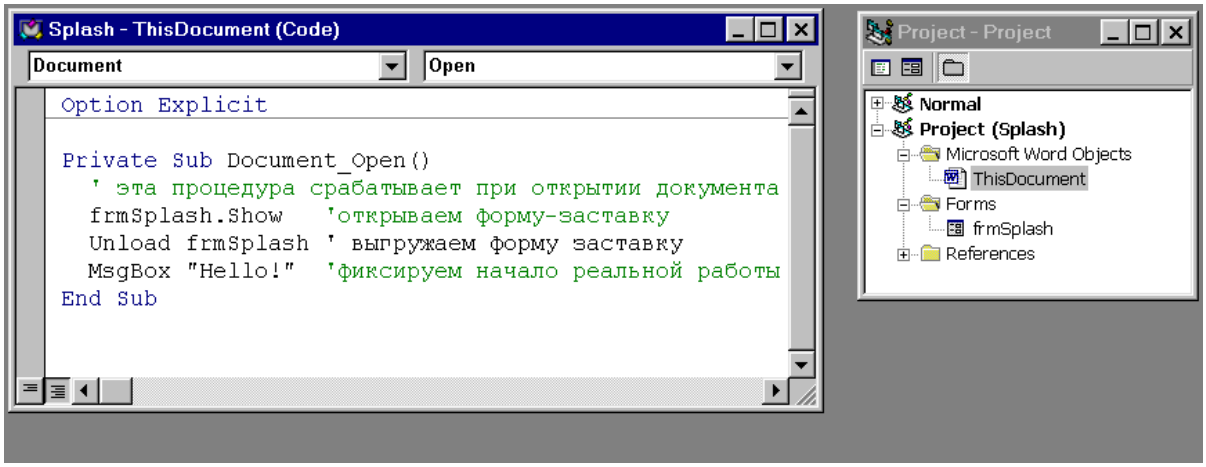


Рис. 247-2 Для проверки работы этой процедуры нужно сначала сохранить и закрыть документ, а затем заново открыть его.

Совет: *Чтобы избежать постоянного закрытия-открытия документа для проведения отладки, измените в описателе Sub Document_Open ключевое слово Private на Public. В этом случае данная процедура автоматически попадет в список макрокоманд документа, и вы сможете обращаться к ней любой момент без перезагрузки документа.*

Однако, запустив процедуру Document_Open, мы обнаружим, что характер ее работы отличается от того, как она функционирует в обычном Visual Basic: здесь форма- заставка не исчезает до тех пор, пока пользователь сам не закроет ее.

Причина такого поведения заключается в том, что форма может открываться в двух разных режимах: модальном и немодальном. В первом случае управление полностью передается форме (остальные элементы интерфейса становятся недоступными пользователю) и возвращается обратно в вызывающую программу только после закрытия этой формы. В немодальном режиме форма открывается и управление сразу возвращается в программу (форма начинает работать как бы в автономном режиме, параллельно со всеми остальными элементами программы).

Различие в поведении формы-заставки в VB и VBA заключается в том, что благодаря используемому по умолчанию параметру метода Show в первом случае форма открывается в немодальном режиме, а во втором — в модальном.

Более того, в Office 97 был реализован только модальный режим работы форм. Поэтому для автоматического удаления формы-заставки нужно придумывать какой- нибудь программный трюк внутри ее модуля, который бы, например, отслеживал некую временную задержку и сам закрывал форму. В Office 2000 появилась возможность использования немодальных форм, но для этого нужно указать параметр Show в явном виде:

```
frmSplash.Show 0   ' открывает форму в немодальном режиме
```

Как нам сделать, чтобы форма-заставка не мелькала на экране, а оставалась некоторое минимальное время, например 5 секунд? Для этого можно воспользоваться подпрограммой временной задержки SleepVB, приведенной в Совете 246, и тогда код открытия документа будет выглядеть следующим образом:

```
frmSplash.Show 0   ' открываем форму-заставку
' Call AllStartOperations  необходимые дополнительные операции
Call SleepVB (5)  ' задержка на 5 секунд
Unload frmSplash  ' выгружаем форму-заставку
Msgbox "Hello!"   ' фиксируем начало реальной работы
```

В этом варианте видны два недостатка. Во-первых, если дополнительные операции будут занимать 20 секунд, то к ним все равно добавятся дополнительные 10 секунд, что будет уже лишним. Во-вторых, пользователь никак не сможет повлиять на сокращение этой временной задержки, например, самостоятельно закрыв форму-заставку.

Для устранения этих недочетов нам придется написать программный код временной задержки непосредственно внутри нашей стартовой процедуры:

```
frmSplash.Show 0   ' открываем форму-заставку
```

```
'
PauseTime = 10 ' задержка в 10 секунд
Start = Timer
' Call AllStartOperations ' необходимые дополнительные операции
Do While Timer < Start + PauseTime
  ' обеспечивает параллельное выполнение других процессов
  DoEvents
Loop
'
Unload frmSplash  ' выгружаем форму-заставку
Msgbox "Hello!"   ' фиксируем начало реальной работы
```

Обратите внимание, что здесь мы вставили обращение к процедуре AllStartOperations после получения начального времени таймера. Таким образом, время появления заставки на экране будет оптимизировано с учетом длительности выполнения дополнительных операций, то есть мы устранили первый недочет начального варианта.

Для понимания смысла функции DoEvents (см. Совет 246[30]) временно заблокируйте обращение к ней, поставив знак комментария. Запустите, процедуру на выполнение и вы увидите, что форма-заставка появляется без рисунка!

Объясняется это тем, что управление возвращается в вызывающую программу до полного завершения прорисовки формы, но этот процесс не может завершиться из-за выполнения цикла Do... Loop. Функция же DoEvents обеспечивает возможность выполнения других параллельных процессов, в том числе и завершения прорисовки заставки.

Как нам теперь дать возможность пользователю сократить время задержки, например, закрыв форму-заставку щелчком мыши? Тут могут быть два варианта в реализации цикла ожидания.

1. Анализ числа открытых форм:

```
2.   Do
3.     OpenForms = DoEvents    ' число открытых форм в приложении
4.   Loop While Timer < Start + PauseTime And OpenForms > 0
```

   Но этот вариант не очень хорош, так как к этому моменту могут быть открыты еще какие-либо другие формы приложения (в процедуре StartAllOperations). Поэтому лучше воспользоваться другим способом.

5. Анализ состояния формы-заставки:

```
6.   Do
7.     DoEvents
8.   Loop While Timer < Start + PauseTime And frmSplash.Visible
```

Вот, собственно, и все по поводу создания стартовых заставок. Обратите внимание, что реализация этих программных конструкций в VB и VBA практически одинакова.

Совет 248. Как изучать программирование в среде VBA

Парадокс заключается в том, что освоение программирования в среде MS Office представляет определенные трудности как для начинающих программистов, так и для опытных разработчиков. Первым нужно изучить незнакомый мир программирования, а вторым — овладеть функциональными возможностями офисных пакетов. И как

[30] http://www.visual.2000.ru/develop/ms-vb/tips/0002-2.htm

выясняется на практике, для этого нужно читать не только книги по "программированию в Office", но и литературу (уже давно опубликованную) по собственно самому программированию и работе с офисными пакетами.

Не менее половины вопросов по "MS Office как средство разработки", поступающих в российский центр технической поддержки, связано с темой "как научиться программировать": как перекодировать данные, как работать со строковыми переменными, как использовать базы данных и пр.

Ответ на подобные вопросы довольно прост: изучайте материалы по программированию в среде обычного VB. Что касается собственно самого программирования, MS Office и VB пересекаются на 70-80 процентов. И не нужно удивляться, что в книгах по "Office/VBA" вопросы технологии написания кода зачастую освещаются довольно скупо. Просто эти темы рассматриваются более подробно в других книгах.

Информацию об информационных ресурсах по VB можно найти по адресу www.visual.2000.ru/develop/

Совет 249. Как обрабатывать строки на VBA и каковы преимущества для этого в MS Office 2000

Данный вопрос как раз относится к категории "как изучать программирование". Строки в VBA обрабатываются точно так же, как и в обычном VB. "Обработка строк" рассматривается во многих книгах, но, как правило, это делается довольно бегло, в рамках отдельной главы.

Современные авторы не уделяют особого внимания этому вопросу, считая, что данная тема была досконально описана еще во времена Basic/DOS. Поэтому советуем заглянуть в архив, находящийся по адресу www.visual.2000.ru/develop/basicdos[31]. Из современных публикаций рекомендуем прочитать соответствующую главу в книге "Использование Visual Basic 6.0"[32], а также статью "Особенности работы со строковыми переменными в Visual Basic"[33], опубликованную в журнале "КомпьютерПресс" NN 10/99 и 01/2000.

Совет 250. Как перекодировать почтовые сообщения

Проблемы с двойной перекодировкой почтовых сообщений встречаются не часто, но они все же существуют.

В библиотеке Office Extensions[34] имеется довольно большая группа решений этой проблемы. Дело в том, что включить макрос в Outlook 97/98 было не очень просто (там использовался механизм VBScript, а не VBA). Поэтому большинство решений основывалось на варианте, когда исходный текст копируется в среду Word, где и производится перекодирование. К счастью, Outlook 2000 содержит уже полнофункциональный VBA.

Если вы хотите написать собственный макрос перекодирования текста, то советуем посмотреть решение RusTextC в библиотеке Office Extensions, которое представлено в

[31] http://www.visual.2000.ru/develop/basicdos/index
[32] http://www.visual.2000.ru/develop/ms-vb/tips/www.visual.2000.ru/develop/vb/books-vb/index.htm
[33] http://www.visual.2000.ru/develop/ms-vb/tips/www.microsoft.ru/msdn/library/kolesov/
[34] http://www.microsoft.ru/offext/

виде "открытого кода". А то, как интегрировать макрос в среду Word, можно прочитать в статье "Разработка приложений в среде Office 97"[35].

Андрей Колесов, Ольга Павлова

© 2000, Андрей Колесов, Ольга Павлова
Авторский вариант. Статья была опубликована с незначительной литературной правкой в журнале "КомпьютерПресс" N 3/2000, компакт-диск.

2^8-1 = 255 (десятичная) = 377 (восьмеричная) = FF (шестнадцатеричная) = B11111111(двоичная)

Ровно четыре года назад в КомпьютерПресс N 3'96 была опубликована наша первая статья под названием "Советы для тех, кто программирует на Visual Basic". В ней было всего три совета, но, честно говоря, в тот момент мы не думали, что их число будет расти и к сегодняшнему дню достигнет предела значения байтовой переменной — 255.

Но мы надеемся, что проблем с переполнением счетчика не будет: двухбайтная Integer обеспечит еще несколько лет для счета советов, а там можно будет перейти и к Long.

Кстати, появление этого цикла статей именно в КомпьютерПресс было совсем не случайным: еще за четыре года до этого, в 1991 году, также в мартовском номере журнала появилась статья "QuickBASIC — это то, что вам нужно", которая стала нашей первой публикацией в только зарождавшейся тогда отечественной компьютерной прессе.

Совет 251. Как копировать ячейки между рабочими книгами

Наш читатель прислал письмо, в котором рассказывает о следующей проблеме. Необходимо программным образом скопировать содержимое ячеек из таблицы одной рабочей книги в таблицы другой. Для этого он написал такую макрокоманду в рабочей книге Macros.xls:

```
Sub Макро1()
    Workbooks.Open Filename:="Source.xls" ' открываем книгу Source.xls
    Cells.Select       ' выделяем все ячейки в активной таблице
    Selection.Copy                        ' копируем в буфер обмена
    ActiveWindow.Close                    ' закрываем активную таблицу
    ' теперь активной стала текущая таблица книги Macros.xls
    Range("A1").Select                    ' начальная позиция для вставки
    ActiveSheet.Paste                     ' вставка из буфера обмена
End Sub
```

Однако результат выполнения копирования из таблицы книги Source.xls с помощью этой макрокоманды отличался от ожидаемого как в среде Excel 97, так и в Excel 2000 (с помощью команд в среде пакета эта операция выполняется верно):

[35] http://www.microsoft.ru/offext/officedev/articles/kolesov/

	Содержимое ячеек			
Исходное содержимое в книге Source	12,12	5,559	5,7777	5.559
Результат после выполнения Макро1 (Excel 97)	12,12	5 559	57 777	5,559
Результат после выполнения Макро1 (Excel 2000)	12,12	5,559	5,7777	5,559

Из этих данных видно, что ошибка копирования каким-то образом связана с неверным использованием региональных установок (хотя работа велась в среде русскоязычных Windows и Office) — очевидна путаница русских и английских установок в Excel 97.

Так 12,12 с точки зрения RegionalSetting = 1033 (США) является просто символьной строкой. И в данном случае она копируется как символьная строка (обратите внимание, что после выполнения Макро1 в Excel 97 ячейка стала выровнена по левому краю). А 5,559 представляет (для установок США) целое число с точкой в качестве разделителя триад. При вставке же этого числа используется русский разделитель триад - пробел. Нечто аналогичное, но плохо поддающееся логическому объяснению, происходит с числом 5,7777. Содержимое же четвертой ячейки — 5.559 — в Source.xls является символьной строкой, выровненной по левому краю. Но для установок США это число с десятичной точкой, которая в русской языке меняется на запятую (и, соответственно, выравнивается по правому краю). Эта же ошибка имеет место в Excel 2000.

Механизм ошибки понятен, но что же делать? Как копировать ячейки?

По нашей просьбе служба технической поддержки в России занялась этой проблемой и после запроса в европейский центр получила такой ответ:

"Переменная копируется как текстовая строка в буфер обмена, после чего исходная книга закрывается и выполняется вставка во вторую книгу. Excel старается преобразовать ее в вид по своему усмотрению и делает это неверно.

Но все будет работать нормально, если держать исходную рабочую книгу открытой до окончания вставки данных".

И действительно, все работает без ошибок, если использовать для копирования такой вариант макроса:

```
Sub Макро2()
    Workbooks.Open FileName:="c:\Source.xls"
    Cells.Select
    Selection.Copy
    ' делаем активной книгу Macros.xls
    Workbooks("Macros.xls").Activate
    Range("A1").Select
    ActiveSheet.Paste ' вставка
    ' только теперь закрываем исходную книгу
    Workbooks("Source.xls").Close
End Sub
```

Еще один совет из службы поддержки Microsoft: без особой нужды не следует копировать таблицу целиком — используйте только тот диапазон, который вам действительно нужен. То есть вместо

```
Cells.Select
```

в данном случае лучше написать:

```
Range("A1:E2").Select
```

При тестировании прилагаемых программных примеров необходимо файл Source.xls скопировать в каталог C:\ или указать другой путь к файлу в коде макросов.

Совет 252. Как прервать вычислительный процесс

Если вы создаете программы с достаточно длительными вычислительными процедурами, то бывает очень полезным обеспечить возможность аварийного прерывания этих процессов. Например, щелкнуть мышью на кнопке в форме. Рассмотрим один из вариантов решения этой задачи.

Предположим, у вас в модуле Process.bas есть некая вычислительная процедура. Идея реализации механизма ее прерывания заключается в использовании некоторого флага (глобальная переменная), состояние которого отслеживается внутри процедуры и который может изменяться другой внешней процедурой.

В этом случае модуль Process.bas может выглядеть приблизительно так:

```
Global ProcessFlag% ' флаг управления процессом, глобальная переменная

Public Sub Process(Result%)
    ' имитация некоего процесса
    ' Result возвращает флаг окончания процесса
    ' 1 - закончился сам, 0 - внешнее аварийное завершение
    '
    ProcessFlag = 1 ' флаг начала процесса
    For i& = 1 To 1000000
        ' увеличить длину цикла, если слишком малое время задержки
        If ProcessFlag = 0 Then Exit For 'проверка флага
        Value# = 100 / 0.3 * 1.5 / 2.3
    Next
    Result = ProcessFlag
End Sub
```

Теперь создадим форму frmInterrupt, на которой поместим кнопку с названием "Щелкни здесь, чтобы прервать процесс Process". Для кнопки напишем такую процедуру:

```
Private Sub Command1_Click()
    ProcessFlag = 0 ' очистка глобального флага
End Sub
```

Далее нужно написать главную процедуру нашего проекта в отдельном модуле MainSub.bas:

```
Public Sub Main()
    ' процедура для демонстрации механизма прерывания
    ' некоего вычислительного процесса
```

```
    frmInterrupt.Show 0   ' открываем форму в немодальном режиме
    Call Process(Result%) ' запускаем форму
    Unload frmInterrupt   ' выгружаем форму
    ' анализ кода завершения процедуры
    MsgBox "Результат завершения процедуры = " & Result%
End Sub
```

Внешне все выглядит правильно, но, запустив проект на выполнение, мы обнаружим, что произвести прерывание процесса не удается. Более того, даже форма не прорисовывается до конца. Подобный вопрос мы обсуждали месяц назад в советах 246-247: дело в том, что вычислительный цикл съедает все ресурсы компьютера, не давая выполняться другим процессам приложения. Для решения этой проблемы необходимо внутрь вычислительного цикла вставить функцию DoEvents, которая передает управление операционной системе для выполнения других процессов.

Однако вариант

```
For i = 1 To 1000000
    DoEvents ' передача управления другим процессам
    If ProcessFlag = 0 Then Exit For 'проверка флага
    Value# = 100 / 0.3 * 1.5 / 2.3
Next
```

также не очень хорош - выполнение DoEvents требует очень много времени. В нашем простом вычислительном примере мы увидим, что быстродействие процедуры Process упадет в 1000 раз (то есть время выполнения DoEvents в 1000 раз больше, чем полезное вычисление по формуле). Чтобы избежать таких непроизводительных затрат, можно написать следующий код:

```
For i = 1 To 1000000
    If i Mod 100000 Then   ' проверка один раз на 100 000 циклов
        DoEvents           ' передача управления другим процессам
        If ProcessFlag = 0 Then Exit For ' проверка флага
    End If
    Value# = 100 / 0.3 * 1.5 / 2.3
Next
```

В этом случае DoEvents будет занимать лишь 1% от полезных вычислений. К сожалению, такой вариант решения задачи отслеживания двух параллельных процессов представляется не самым лучшим, но таковы уж реалии Windows.

Короче говоря, совет такой: для обеспечения прерывания вычислительных задач по ходу программы расставляйте (но не очень часто) подобные конструкции:

```
  DoEvents
  If ProcessFlag = 0 Then Exit something
```

Совет 253. Как решить проблему с сохранением проектов с цифровой подписью

В ряде случаев в Office 2000 возникает ситуация, когда пользователь не может сохранить проект (документ с макрокодом) с электронной подписью - запись документа возможна только без подписи. При этом выдается неверная диагностика о нехватке места на диске.

В данном случае имеет место программная ошибка, которая, как мы надеемся, будет исправлена. На самом же деле она проявляется только в тех проектах, которые содержат внутренние синтаксические ошибки в коде. Поэтому, встретив подобное сообщение о невозможности сохранения документа с электронной подписью, мы рекомендуем вам, прежде чем отменять подпись, проверить работоспособность вашего кода. Довольно часто ошибка связана с отсутствием описания переменной или ссылки на внешний объект. Чтобы лучше понять суть ситуации, сделайте такой простой пример в Word. Создайте новый документ и перейдите в среду VBA. Там создайте макрокоманду Test1:

```
Sub Test1 ()
    Avar = 1
End If
```

Разумеется, сначала должен быть задан режим Option Explicit (обязательное объявление переменных).

Теперь установите электронную подпись и попробуйте сохранить документ. Скорее всего, у вас появится сообщение о нехватке места на диске для записи файла.

Запустите макрокоманду Test1 на выполнение - транслятор выдаст сообщение о синтаксической ошибке (не определена переменная Avar). Добавьте в процедуру описание:

```
Dim Avar As Integer
```

Теперь макрокоманда станет выполняться (правда, не делая ничего полезного), и документ тоже без проблем сохранится с электронной подписью.

Совет 254. Как автоматически определить кодовую таблицу для русских текстов

Проблема известна: для русского языка применяется несколько различных кодовых таблиц, и поэтому актуальной задачей является преобразование текстов из одной кодировки в другую. Для чего нужно автоматически определять используемую кодовую таблицу? Тут есть два очевидных примера:

1. Это необходимо при загрузке Web-страниц в браузер, текстовых файлов — в Word или при работе с почтовыми программами (гораздо лучше поступать так, чем заниматься перебором разных вариантов кодировок).
2. Это нужно для дополнительного контроля при преобразовании кодов файлов. Например, мы постоянно осуществляем преобразование HTML-страниц из Windows в KOI8 (наш Web-сервер работает под UNIX), но при этом порой из-за невнимательности либо делаем двойную перекодировку, либо неправильно устанавливаем исходный код, либо вообще пропускаем файлы.

Идея автоматического определения кодировки русских текстов достаточно очевидна: необходимо определить частоту попадания кодов 128-255 (&h80-&hFF) в различные числовые диапазоны. Понятно, что основное количество этих кодов приходится именно на русские буквы (в старшей половине кодовой таблицы находится также ряд специальных символов типа "№", открывающих-закрывающих кавычек, Copyright и др.), причем строчных букв гораздо больше, чем прописных.

В таблице 254 приведены результаты подсчета такой статистики для довольно типичного русскоязычного текста, которые получены с помощью подпрограммы CodeTableTest (листинг 254[36]).

Таблица 254. Анализ частоты распределения кодов 128-255 для типичного русскоязычного текста для разлиных кодовых таблиц

Диапазон кодов (шестнабцатиричная система)	**Процент попадания кода в соответствующий диапазон**				
	Windows, cp1251	**KOI8-R**	DOS, cp866 (альтернативная)	**ISO8859-5 (основная)**	**Macintosh**
80-8F	0,00	0,00	1,24	0,00	**1,24**
90-9F	0,20	0,20	0,63	0,20	0,43
A0-AF	0,01	0,01	**67,96**	0,01	0,01
B0-BF	0,01	0,01	0,01	1,24	0,01
C0-CF	1,24	**61,3**	0,00	0,43	**0,00**
D0-DF	0,43	**36,8**	0,00	**67,96**	2,28
E0-EF	**67,9**	0,80	**30,16**	**30,16**	**67,9**
F0-FF	**30,2**	0,8	0,01	0,01	**28,08**

Тут хорошо видно, что строчные буквы попадают в разные числовые диапазоны для разных кодовых таблиц. Любопытно также отметить, что частота появления букв первой половины русского алфавита (от "а" до "п") в два раза выше, чем частота букв от "р" до "я". Соответственно, если принять за условие, что процент строчных русских букв среди кодов 128-255 превышает заданную величину, например, Lpercent = 0,70, то критерий определения кодовой таблицы будет выглядеть таким образом:

Windows	частота (&hE0-&hFF) > Lpercent
KOI8-R	частота (&hC0-&hDF) > Lpercent
DOS	частота (&hA0-&hAF + &hE0-&EF) > Lpercent
ISO	частота (&hD0-&hEF) > Lpercent

Единственная проблема здесь заключается в идентификации различных кодировок Windows и Macintosh, у которых диапазоны кодов русских строчных букв почти совпадают. Но тут можно осуществить дополнительную проверку, которая основана на том, что в Windows практически не используются коды в диапазоне &h80-&h8F, а прописные буквы от "А" до "П" (&hC0-&CF) составляют значительную величину, тогда как в Macintosh ситуация для этих же диапазонов диаметрально противоположная.

[36] http://www.visual.2000.ru/develop/ms-vb/tips/0003.htm

Функция NumberTableTest, реализующая описанный выше алгоритм, приведена в листинге 254[37].

Обратите внимание еще на один важный аспект — для хранения кодов символов мы используем не строковую переменную, а байтовый массив. Соответственно, если нужно прочитать исходный файл, необходимо применять следующую конструкцию:

```
Open SourceFile$ For Binary As #1
    LenS = LOF(1)
    ReDim bytSourceArray(1 To LenS) As Byte
    Get #1, , bytSourceArray
Close #1
```

Если же вам нужно преобразовать данные, изначально представленные в виде строковой переменной, то следует выполнить такое преобразование: BytSourceArray() = StrConv (SourceString$, vbFromUniCode, &h419)

Листинг 254. Процедуры CodeTableTest и NumberTableTest для совета 254

```
Public Sub CodeTableTest _
    (bytSourceArray() As Byte, TableTest() As Single)
    '
    ' Вычисление частоты различных кодов
    ' (для старшей части таблицы 128-255)
    '
    ' ВХОД: bytSourceArray() - байтовый массив кодов символов
    ' ВЫХОД: TableTest(-1) - общее число символов
    ' (без учета кодов перевода строки)
    ' TableTest(0) - общее число кодов 128-255
    ' TableTest(1) - частота в диапазоне &H80-&H8F
    ' ...
    ' TableTest(8) - частота в диапазоне &HF0-&HFF
    '==================================
    Dim i%, k&, Code%
    For i = 0 To 8: TableTest(i) = 0: Next
    For k = LBound(bytSourceArray) To UBound(bytSourceArray)
        Code = bytSourceArray(k)
        If Code <> 10 And Code <> 13 Then
            TableTest(-1) = TableTest(-1) + 1
        End If
        If Code > 127 Then
            TableTest(0) = TableTest(0) + 1
            i = (Code - 128) \ 16 + 1
            TableTest(i) = TableTest(i) + 1
        End If
    Next
    For i = 1 To 8:
        TableTest(i) = TableTest(i) / TableTest(0)
    Next
End Sub

Public Function NumberTableTest _
    (TableTest() As Single, Lpercent As Single) As Integer
    '
    ' Определение номера кодовой таблицы
    ' ВХОД: TableTest () - см. описание процедуры CodeTableTest
    ' Lpercent - минимальная доля строчных русских букв
    ' ВЫХОД: NumberTableTest = 0 - неопределено
```

[37] http://www.visual.2000.ru/develop/ms-vb/tips/0003.htm

```
    ' = 1 - DOS (cp866)
    ' = 2 - Windows (cp1251)
    ' = 3 - UNIX (KOI-8)
    ' = 4 - General(ISO 8859-5)
    ' = 5 - Macintosh

    '====================================
    NumberTableTest = 0 ' неопределена

    If TableTest(7) + TableTest(8) > Lpercent Then
        If TableTest(1) > TableTest(5) Then
            NumberTableTest = 5 ' Macintosh
        Else
            NumberTableTest = 2 ' Windows, cp1251
        End If
    ElseIf TableTest(5) + TableTest(6) > Lpercent Then
        NumberTableTest = 3 ' KOI8 (Unix)
    ElseIf TableTest(7) + TableTest(3) > Lpercent Then
        NumberTableTest = 1 ' DOS (cp866)
    ElseIf TableTest(6) + TableTest(7) > Lpercent Then
        NumberTableTest = 4 ' ISO 8859-5/General
    End If
End Function
```

Совет 255. Как определить кодировку текста: еще один вариант

Алгоритм, приведенный в предыдущем совете, не работает в случае неверной перекодировки текста, то есть когда исходные данные уже не соответствуют ни одной из кодовых таблиц. (Когда, например, текст, записанный в Windows, ошибочно преобразуется "из DOS в любой другой".) В таких случаях восстановление текста порой бывает вообще невозможно, но в любом случае подобную ситуацию полезно идентифицировать.

С проблемой перекодировки мы чаще всего сталкиваемся при создании собственных HTML-страниц, предназначенных для размещения на Web-сервере. Для контроля реальной кодовой таблицы в данном файле можно использовать следующий прием. (Мы сами выполняем сканирование обновлений локального варианта нашего сервера перед их записью на удаленный компьютер в целях обнаружения подобных ошибок.)

В каждую новую HTML-страницу вставляется строка комментария, в которую включены все буквы русского алфавита, кроме "ё" (такая строка автоматически создается нашим простеньким генератором страниц):

```
<!CodePage=А...Яа...я>
```

Конечно, можно минимизировать число используемых символов, но полный их набор точно гарантирует нужное решение.

Программный код утилиты TstCode2, которая проверяет код отдельного файла, приведен в листинге 255[38]. Ключевой процедурой здесь является NumberTableTestKey, выполняющая идентификацию символьного кода. В свой реальной работе для проверки кодовой таблицы текстовых и HTML-файлов мы пользуемся утилитой TestCode (рис. 255), которая применяет комбинацию двух описанных выше методов.

[38] http://www.visual.2000.ru/develop/ms-vb/tips/0003.htm

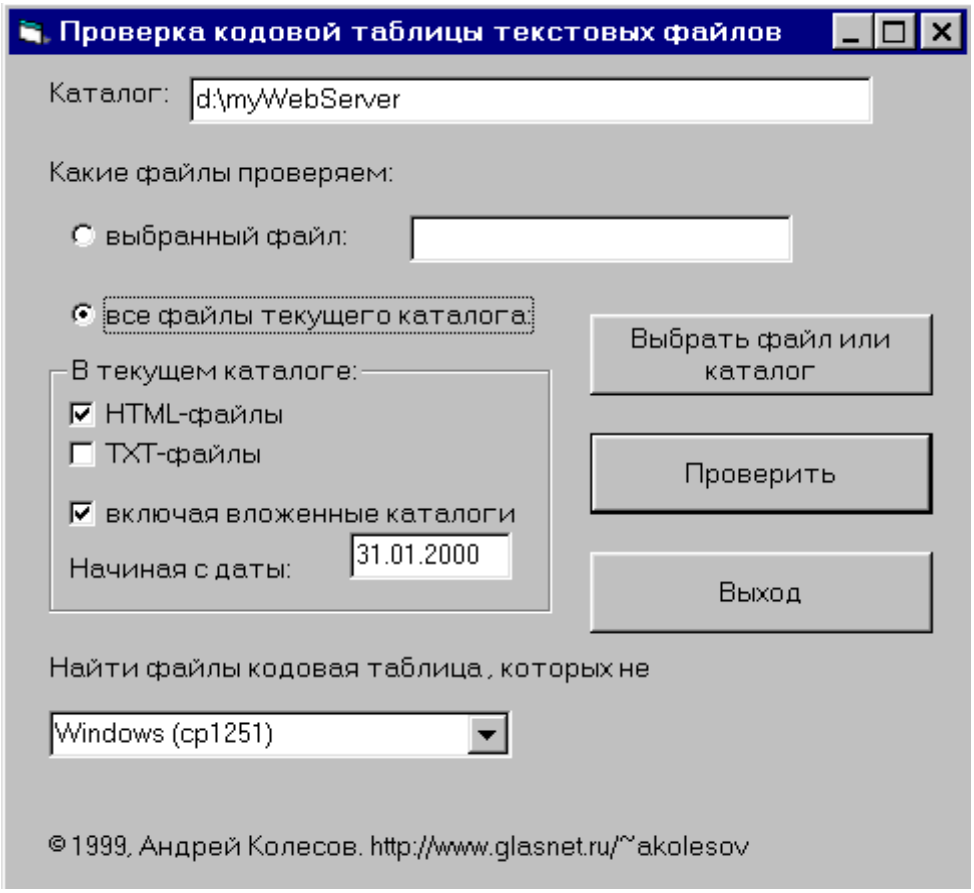


Рис. 255

Листинг 255. Код утилиты RusCode2 для Совета 255

```
Public Sub Main()
    ' Стартовая процедура
    ' для утилиты RusCode2 -
    ' определение кодовой таблицы текста по ключевому параметру
    '==========================
    Const KeyCode$ = "!Codepage=" ' ключ поиска
    Const Password$ = "Кодовая таблица" ' идентификатор

    Dim SourceFile$, LenS&, NumByte&, Source$

    Dim TableName$(-1 To 5) ' названия таблиц
    TableName$(-1) = "Нет ключа-идентификатора"
    TableName$(0) = "Неопределена"
    TableName$(1) = "DOS (cp866)"
    TableName$(2) = "Windows (cp1251)"
    TableName$(3) = "UNIX (KOI-8)"
    TableName$(4) = "General(ISO 8859-5)"
    TableName$(5) = "Mac"

    SourceFile$ = App.Path + "\" + "TestWin.htm"
    ' чтение исходного текстового файла
    Open SourceFile$ For Binary As #1
        LenS = LOF(1)
        Source$ = Space$(LenS)
        Get #1, , Source$
    Close #1
    ' поиск ключа
    NumByte = NumberTableTestKey(Source$, KeyCode$, Password$)
```

```
    MsgBox "Кодовая таблица файла " & vbCrLf & _
        SourceFile$ & vbCrLf & _
        "= " & TableName$(NumByte)
End Sub

Public Function NumberTableTestKey%(Source$, KeyCode$, Password$)
    ' Проверка кодовой таблицы по ключевым параметрам
    'ВХОД: Source$ - содержимое исходного файла
    ' KeyCode$ - ключ для поиска
    ' Password$ - идентификатор
    'ВЫХОД: значение функции (см. содержимое TableName())
    '
    Dim NumByte&, StrWord$, i%

    NumByte = InStr(Source$, KeyCode$)
    If NumByte <= 0 Then ' нет записи с ключом
        NumberTableTestKey = -1
        Exit Function
    End If

    'выделяем идентификатор
    StrWord$ = Mid$(Source$, NumByte + Len(KeyCode$),     Len(Password$))

    If StrWord$ = Password$ Then 'Windows, cp1251
    NumberTableTestKey = 2: Exit Function
    End If
    NumByte = 0
    For i = 1 To 5 ' сравнение перебором
        If i <> 2 Then '
            If StrWord$ = RusDosOther$(Password$, 2, i) Then
                NumByte = i: Exit For
            End If
        End If
    Next
    NumberTableTestKey = NumByte
End Function
```

Все программные примеры, которые использовались в приведенных здесь советах, можно найти по адресу: www.visual.2000.ru/develop/vb/source/.

Андрей Колесов, Ольга Павлова

© 2000, Андрей Колесов, Ольга Павлова
Авторский вариант. Статья была опубликована с незначительной литературной правкой в журнале "КомпьютерПресс" N 4/2000, компакт-диск.

Совет 256. Программное обращение к CommonDialog

Если требуется выполнить выборку имени файла для операций открытия или закрытия, для этого обычно используется элемент управления CommonDialog. Однако в ряде случаев полезнее бывает выполнить прямое обращение к функциям GetOpenFileName и GetSaveFileName библиотеки COMDGL32.DLL. Например, если вы пишете собственный мастер по созданию проекта, то подключиться в проект с модулем кода гораздо удобнее, чем делать коррекцию кода модуля формы. Тем более что такая процедура будет единой для всего проекта.

В листинге 256[39] приведен один из вариантов реализации универсальной процедуры FileOpenSave поиска имени файла в режиме "Открыть/Сохранить", а также пример обращения к ней.

Листинг 256. Обращение к диалоговому окну "Открыть/Сохранить" через WinAPI

```
Private Declare Function GetOpenFileName Lib "comdlg32.dll" _
    Alias "GetOpenFileNameA" (ofn As OPENFILENAME) As Boolean
Private Declare Function GetSaveFileName Lib "comdlg32.dll" _
    Alias "GetSaveFileNameA" (ofn As OPENFILENAME) As Boolean
' Описатель данных для работы с окном "Открыть/Сохранить файл"
Private Type OPENFILENAME
    lStructSize As Long
    hwndOwner As Long
    hInstance As Long
    stFilter As String
    stCustomFilter As String
    nMaxCustFilter As Long
    nFilterIndex As Long
    strFile As String
    nMaxFile As Long
    stFileTitle As String
    nMaxFileTitle As Long
    stInitialDir As String
    strTitle As String
    Flags As Long
    nFileOffset As Integer
    nFileExtension As Integer
    stDefExt As String
    lCustData As Long
    lpfnHook As Long
    lpTemplateName As String
End Type
' Константы для управления флагами окна "Открыть/Сохранить файл"
Public Enum OFN_FLAGS
    OFN_READONLY = &H1
    OFN_OVERWRITEPROMPT = &H2
    OFN_HIDEREADONLY = &H4
    OFN_NOCHANGEDIR = &H8
    OFN_SHOWHELP = &H10
    OFN_ENABLEHOOK = &H20
    OFN_ENABLETEMPLATE = &H40
    OFN_ENABLETEMPLATEHANDLE = &H80
    OFN_NOVALIDATE = &H100
    OFN_ALLOWMULTISELECT = &H200
    OFN_EXTENSIONDIFFERENT = &H400
    OFN_PATHMUSTEXIST = &H800
    OFN_FILEMUSTEXIST = &H1000
    OFN_CREATEPROMPT = &H2000
    OFN_SHAREAWARE = &H4000
    OFN_NOREADONLYRETURN = &H8000
    OFN_NOTESTFILECREATE = &H10000
    OFN_NONETWORKBUTTON = &H20000
    OFN_NOLONGNAMES = &H40000
    OFN_EXPLORER = &H80000
    OFN_NODEREFERENCELINKS = &H100000
    OFN_LONGNAMES = &H200000
End Enum
```

[39] http://www.visual.2000.ru/develop/ms-vb/tips/0004-1.htm

```
Public Function FileOpenSave( _
  Optional ByVal OpenFile As Boolean = True, _
  Optional ByRef Flags As Long = 0&, _
  Optional ByVal InitialDir As Variant, _
  Optional ByVal Filter As String = vbNullString, _
  Optional ByVal FilterIndex As Long = 1, _
  Optional ByVal DefaultExt As String = vbNullString, _
  Optional ByVal FileName As String = vbNullString, _
  Optional ByVal DialogTitle As String = vbNullString, _
  Optional ByVal hwnd As Long = -1) _
  As String
  '
  ' Процедура обращения к
  ' диалоговому окну "Открыть/Закрыть файл"
  ' OpenFile = True    - ОТКРЫТЬ (по умолчанию)
  '                    = False - ЗАКРЫТЬ
  '
  Dim ofn As OPENFILENAME    ' структура для обращения
                  ' к DLL-функции
  Dim stFileName As String
  Dim stFileTitle As String
  Dim fResult As Boolean
  ' начальный каталог
  If IsMissing(InitialDir) Then InitialDir = CurDir
  If (hwnd = -1) Then hwnd = 0    ' установка описателя

  ' Подготовка строковых переменных
  stFileName = Left$(FileName & String$(256, vbNullChar), 256)
  stFileTitle = String$(256, vbNullChar)

  ' формирование данных для обращения к окну
  With ofn
    lStructSize = Len(ofn)
    hwndOwner = hwnd
    stFilter = Filter
    nFilterIndex = FilterIndex
    strFile = stFileName
    nMaxFile = Len(stFileName)
    stFileTitle = stFileTitle
    nMaxFileTitle = Len(stFileTitle)
    strTitle = DialogTitle
    Flags = Flags
    stDefExt = DefaultExt
    stInitialDir = InitialDir
    hInstance = 0
    stCustomFilter = String$(255, vbNullChar)
    nMaxCustFilter = 255
    lpfnHook = 0
  End With
  If OpenFile Then    ' открыть файл
    fResult = GetOpenFileName(ofn)
  Else    'сохранить файл
    fResult = GetSaveFileName(ofn)
  End If
  If fResult Then    ' Нажата кнопка "Open/Save"
    Flags = ofn.Flags    ' флаги
    ' имя файла
    FileOpenSave = Left$(ofn.strFile, _
      InStr(ofn.strFile, vbNullChar) - 1)
  Else: FileOpenSave = "" ' нажата Cancel
  End If
End Function
```

```
Public Sub Main()
    ' Тестирование конструкции FileOpenSave:
    ' обращение к окнам "Open/Save File" напрямую через DLL
    '============================

    Dim bOpenFile As Boolean
       ' тип операции — TRUE (открыть)/FALSE (закрыть)
    Dim Filter$, Flags&
    Dim FileName$, InitDir$, Title$

    InitDir$ = App.Path
    Title$ = "Как выбрать имя каталога?"
    tab]Filter$ = "Текстовые файлы (*.txt)" & _
    Chr$(0) & "*.TXT" & Chr$(0) & _
        "Все файлы (*.*)" & Chr$(0) & "*.*" & Chr$(0)
    bOpenFile = True
    FileName$ = FileOpenSave( _
    bOpenFile, , InitDir, Filter, , ".txt", , Title$)
    MsgBox FileName$
End Sub
```

Совет 257. Как узнать, существует ли файл?

Проще всего сделать это следующей операцией в одну строчку:

```
If Dir$(FileName$)="" Then ' такого файла нет!
```

Совет 258. Как выбрать имя каталога

В разных вспомогательных утилитах довольно часто встречается задача выборки имени файла или имени каталога. Например, если вы пишете утилиту по перекодировке текстовых файлов, будет достаточно предоставить пользователю возможность выбрать или конкретный файл, или каталог, в котором нужно преобразовать все файлы.

К сожалению, диалоговое окно Open (или DLL-функция GetOpenFileName) позволяет выбирать только имя файла. Конечно, можно из этого имени по специальной команде пользователя сделать выделение каталога. А что делать, если нужный каталог содержит только подкаталоги?

Разумеется, лучшим вариантом является создание собственного диалогового окна, которое будет учитывать все режимы, необходимые в конкретной задаче. Но можно использовать и стандартное окно Open (рис. 258):

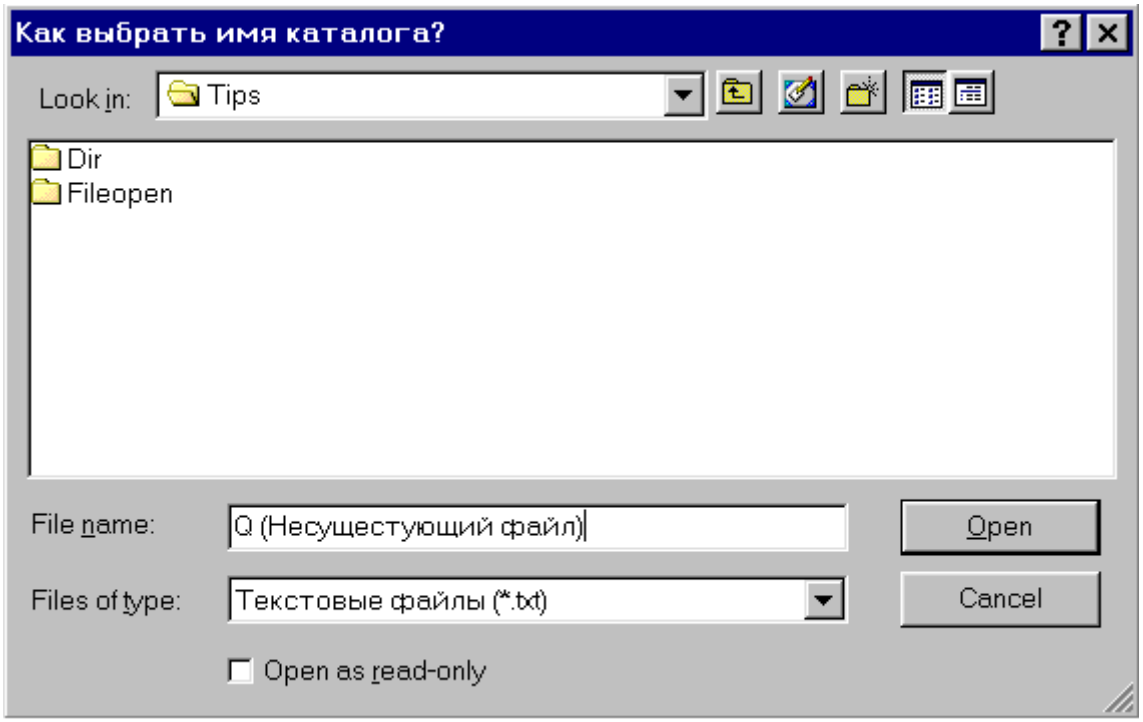


Рис. 258

Для этого в поле File Name нужно просто ввести любое имя несуществующего в данном каталоге файла (например, QQQQ). Соответственно в программе, которая обращается к окну (например, с использованием функции FileOpenSave, о чем говорилось в предыдущем совете), напишется такой программный код:

```
MyFile$ = FileOpenSave$(...)
If Dir$(MyFile$) = "" Then ' задан несуществующий файл
    ' значит задано имя каталога
    In1% = InstrRev (MyFile$, "\")
    MyDir$ = Left$ (MyFile$, in1% - 1) ' имя каталога
    ' обработка данных в режиме "выделен каталог"
End If
```

Совет 259. Какие операторы лучше использовать: Print # или Write #?

В своем совете 228[40] мы привели пример утилиты Summa.vbp (представление числового значения прописью), которая, в частности, сохраняет введенные параметры (для последующего восстановления) в виде простого текстового файла. Данные последовательно записываются оператором Print #, а при запуске утилиты читаются оператором Input #.

В связи с этим наш читатель Константин Абакумов обратил внимание на то, что в справке Visual Basic подчеркивается следующее: *"Для записи данных в файл, который в будущем планируется читать с помощью инструкции Input #, следует вместо инструкции Print # использовать Write #. Это гарантирует, что записанные данные будут корректно разделены и могут быть корректно прочитаны при наличии любых национальных настроек".*

[40] http://www.visual.2000.ru/develop/ms-vb/tips/0001-1.htm

Это замечание совершенно справедливо (хотя конкретно в случае утилиты Summa — не существенно), однако порой Print # является предпочтительнее. Рассмотрим эту ситуацию подробнее.

1. **Преимущества Write #**

   Предположим, вы хотите сохранить четыре переменные — дату, две символьные строки и число:

```
Dim MyDate As Date, MyStr1$, MyStr2$, MyInt%
MyDate = Date$ ' текущая дата
MyStr1 = "Коля"
MyStr2 = "Петя, Лена, 2000"
MyInt = 12
Open MyFile$ For Output As #1
    Print #1, MyDate
    Print #1, MyStr1; MyStr2; MyInt
Close
Open MyFile$ For Output As #1
Input #1, MyDate ' ! здесь будет выдана ошибка!
Input #1, MyStr1; MyStr2; MyInt
Close
```

   Очевидная проблема возникнет при чтении даты, записанной в файл в виде 26.02.00: программа просто не поймет, что это — дата. Можно решить этот вопрос, сделав следующую конструкцию чтения даты:

```
Input #1, MyDate2$ ' чтение строки
MyDate = MyDate2$  ' и преобразование ее в дату
```

   Однако здесь также имеется потенциальная опасность — нужно быть уверенным, что данные читаются и записываются в Windows с одинаковыми региональными установками. (Представьте себе, что вам нужно переслать файл с расчетными данными в виде текстового файла своему коллеге в США.) Еще больше проблем возникнет с конструкцией:

```
Print #1, MyStr1; MyStr2; MyInt
...
Input #1, MyStr1; MyStr2; MyInt
В случае нашего примера вы получите следующие результаты при чтении:
MySrt1 = "КоляПетя"
MySrt2 = "Лена"
MyInt = 2000
```

   Видно, что они отличаются от исходных значений. Если же использовать в операторе Print # в качестве разделителя в списке запятую, то получится следующий результат:

```
MySrt1 = "Коля Петя"
MySrt2 = "Лена"
MyInt = 2000
```

   Если мы заменим оператор Print # на Write #, то все данные будут сохраняться и читаться верно, поскольку дата будет записываться в виде литерала #2000-02-26# (обратите внимание, что литералы даты хранятся в американском формате, независимо от региональных установок), строковые переменные — в двойных кавычках и все поля в текстовом файле будут разделены запятыми.

Но здесь есть очевидная проблема: если ваши строковые переменные содержат двойные кавычки, то будут происходить неприятные искажения данных. Например, вы ввели в текстовое поле название ЗАВОД "САЛЮТ" и хотите сохранить его значение в текстовом файле. Чтобы гарантированно избежать подобной ситуации, нужно проверять строки на наличие двойных кавычек и, в частности, автоматически их удалять (можно поставить соответствующий контроль при вводе данных) или менять на одинарные кавычки.

2. **Преимущества Input #**

Предположим, вам нужно сохранить в виде текстового файла числовой массив с переменными границами Arr (M, N). Конечно, можно воспользоваться оператором Write #:

```
For i = 1 To M
  For j = 1 To N
    Write #1, Arr(i, j)
  Next
Next
```

Однако с таким текстовым файлом будет крайне неудобно работать, если потребуется изучать или корректировать его с помощью простого текстового редактора (такая задача встречается достаточно часто). Файл будет выглядеть гораздо лучше, если применить следующий код:

```
For i = 1 To M
  For j = 1 To N
    Print #1, Arr(i, j); ' запись в одну строку
  Next
  Print #1 ' перевод строки
Next
```

Таким образом, оператор Print # очень полезен в случае, когда нужно вывести в одну строку текстового файла набор данных переменной длины или просто большое число однородных данных.

В свое время мы именно в силу необходимости получения структурируемого текстового файла (хорошо читаемого в текстовом редакторе) полностью отказались от использования оператора Write #. Для беспроблемного применения Print # мы создали очень простой набор процедур, который имитировал Write — заключал в кавычки строки, писал даты в виде литералов, расставлял запятые в качестве разделителей.

Какой вариант удобнее? Это, конечно же, зависит от конкретной решаемой задачи.

Совет 260. Где хранить INI-файл

В данном случае мы имеем в виду файлы, в которых хранятся некоторые параметры приложения между его перезапусками. (В общем случае они могут иметь любое расширение и любую удобную для разработчика структуру.) Конечно, для хранения подобных файлов можно выделить специальный каталог, например системный Windows. Или вообще хранить подобные данные в файле Реестр. Однако, на наш взгляд, такая концентрация данных разных приложений в одном месте является довольно порочной практикой (к сожалению, именно она используется авторами Windows, а по их примеру — и многими независимыми разработчиками). В связи с этим можно привести несколько

возражений: использование единого Реестра снижает производительность и надежность системы ("падение" одного файла приводит к "падению" данных всех приложений), кроме того, резко усложняется проблема "удаления мусора".

Поэтому мы советуем создавать для отдельных приложений собственные INI-файлы. И хранить их лучше всего в одном каталоге с самим приложением — тогда вам не придется писать инструкции пользователям "Создайте каталог IniFolder для хранения файла MyApplication.INI". Более того, может оказаться удобным предоставить возможность использования нескольких вариантов INI-файлов (например, при работе нескольких пользователей). В этом случае вызов утилиты (через ярлык) может выполняться командной строкой:

```
MyUtility [User.INI]
```

Программный код обработки будет в этом случае иметь, например, такой вид:

```
Public Sub Main()
    Dim IniFile$, Say$
    '
    ' Запуск приложения
    ' SomeUtil.EXE [MyIniFile]
    '=====================
    ' открытие INI-файла приложения:
    IniFile$ = Command$ ' читаем командную строку

    If IniFile$ = "" Then ' не задан в командной строке
    ' по умолчанию: имя самой утилиты с расширением INI
        IniFile$ = App.EXEName + ".INI"
    End If
    ' формируем полный путь - там же, где находится приложение
    IniFile$ = App.Path + "\" + IniFile$

    Say$ = "INI-файл = " & IniFile$
    If Dir(IniFile$) <> "" Then
        MsgBox Say$, , "СУЩЕСТВУЕТ"
        ' открытие INI-файла, чтение исходных данных
    Else
        MsgBox Say$, , "НЕ СУЩЕСТВУЕТ"
        ' Тут можно спросить, нужно ли запускать приложение
        ' с параметрами по умолчанию и создавать новый INI-файл
    End If
End Sub
```

При отладке приложений в среде командная строка задается в поле Command Line Arguments в диалоговом окне Project Properties|Make.

Андрей Колесов, Ольга Павлова

© 2000, Андрей Колесов, Ольга Павлова
Авторский вариант. Статья была опубликована с незначительной литературной правкой в журнале "КомпьютерПресс" N 4/2000, компакт-диск.

Совет 261. Как обеспечить совместимость VBA и VB

В статье об интеграции VBA в VB-приложения ("Интеграция VBA в бизнес-приложения независимых разработчиков", КомпьютерПресс N 3/2000) мы отмечали следующую проблему с отладкой приложений.

Если мы запускали созданное приложение (с уже интегрированным в него VBA) непосредственно в среде Visual Basic, то, перейдя затем в VBA, обнаруживали, что глобальный объект приложения CApplication (он создавался мастером VB Integration) недоступен. Если же создать загрузочный EXE-модуль, то с объектом можно будет нормально работать.

Для иллюстрации этой ситуации продемонстрируем несложный пример. При этом имеется в виду, что вы работаете с VB и у вас на компьютере установлен VBA 6.0 SDK 6.1.

1. В среде VB создайте самый простой проект — Standard EXE. Обратите внимание: у нас получилось вполне работоспособное приложение, которое просто загружает формы. Но нам для примера ничего больше и не нужно. Далее запустите VB Integration Wizard из меню Add-ins и пройдите все этапы его работы, просто нажимая кнопку Next (ничего не меняя в полях окон).

   Итак, за полминуты мы создали приложение, которое само по себе ничего не умеет делать, но зато имеет среду VBA, где можно разрабатывать и выполнять автономные VBA-проекты любой сложности.

2. Запустим наше приложение Project1, перейдем в среду VBA (Alt-F11) и напишем в окне Immediate:

```
3.   Print Project1.CApplication.Name
```

   Нажав Enter (выполнив эту строку), мы увидим сообщение: "Method or data member not found".

4. Теперь создадим EXE-модуль нашего проекта и повторим с ним пункт 2. Ошибки не будет, и мы увидим в окне Immediate имя нашего проекта — Project1.

Получается, что отлаживать макрокоманды, которые работают с объектами основного приложения, можно только в варианте EXE. Это, конечно, очень неудобно, особенно в том режиме, когда мы ведем разработку и отладку этих объектов.

Но, оказывается, есть очень простое решение этой проблемы. Нужно просто переименовать объект CApplication в Application, внеся также соответствующие изменения во все упоминания этого объекта, например используя команду Replace. (Еще

проще сделать это в момент работы мастера VB Integration, исправив предлагаемое имя глобального объекта приложения.)

Теперь объект Project1.Application будет доступен и в среде VB, и при работе с EXE-модулем. Почему так получается, что в среде VB доступен только объект с именем Application, — непонятно. Но тем не менее это так!

Совет 262. Как контролировать макросы в шаблонах

В своем обзоре по MS Office 2000 (КомпьютерПресс N 12/99) мы отмечали некоторые проблемы при использовании шаблонов в Word 2000. Так, при загрузке шаблонов в виде автономных документов или присоединенного файла из каталога Шаблоны пользователя не производилась проверка на наличие макрокода, хотя такой режим контроля был задан. Оказывается, проблема заключается в более детальных установках контроля за макрокодом.

Управление режимом контроля за наличием макрокода при загрузке документов производится в диалоговом окне Security[Безопасность] (команда меню Tools|Macro|Securiry [Сервис|Макрос|Безопасность]). На его вкладке Trusted Sources [Надежные источники] имеется флажок Trust all installed Add-ins and templates [Доверять всем установленным надстройкам и шаблонам]. Под "установленными" подразумеваются дополнения и шаблоны, помещенные в каталог "Шаблоны пользователя". (Конкретное имя этого каталога указывается в поле User Templates [Шаблоны пользователя] во вкладке File Locations [Расположение] диалогового окна Tools|Options [Сервис|Параметры].)

По умолчанию данный флажок установлен, поэтому все шаблоны из этого каталога не проверяются на наличие макрокода (подразумевается, что вы помещаете туда файлы, в которых точно уверены). Если вы все же хотите выполнять такую проверку, то очистите флажок.

В Word 97 такого специального режима для загрузки шаблонов не было. Но в начальной версии программы иногда имела место ошибка — когда шаблоны загружались без проверки на макрокод. Этот дефект уже устранен — заплатку, которая решает данную проблему, можно скачать по адресу http://www.microsoft.com/rus/download/wd97sp.htm.

Совет 263. Использование даты в SQL-запросах

В своей статье "Особенности обработки дат в Visual Basic" (КомпьютерПресс N 07'99) мы обращали внимание читателей на необходимость учета национальных особенностей обработки дат. Вот еще один из примеров подобной ситуации.

При обращении к базе данных для выборки каких-то записей по дате нужно написать символьную строку приблизительно такого вида:

```
SELECT * FROM SomeTable WHERE Time <= #02/29/2000#
```

Обратите внимание, что дата задается здесь как литерал, но самое главное — литерал может быть представлен только в формате американской даты. Поэтому вариант, часто предлагаемый американскими авторами:

```
Sql1$ = "SELECT * FROM SomeTable WHERE Time <= "
SQL$ = Sql1$ & "#" & Now & "#"
```

с российскими региональными установками работать не будет. Для этого требуется использовать функцию Format с заданием американского формата:

```
D$ = Format(Now, "MM/dd/yyyy HH:mm")
Mid$(D$, 3) = "/" ' меняем разделители
Mid$(D$, 6) = "/"
SQL$ = Sql1$ & "#" & D$ & "#"
```

Здесь нужно обязательно в явном виде установить разделители в дате, так как, хотя мы и указали "MM/dd/yyyy" в Windows с российскими региональными установками, в символьном представлении даты будут записаны точки.

Совет 264. Программный сброс хранителя экрана

Наш читатель Владимир Чикин прислал нам такой вопрос: каким образом выключить хранитель экрана программным способом из приложения, чтобы вновь был виден программный интерфейс? Это бывает нужно, если приложение работает без использования диалога с пользователем (например, программа отслеживает данные, поступающие из COM-порта, или просто выполняет очень длительные вычисления) и требуется выдать сообщения о каких-либо критических событиях в программе.

Здесь видятся два варианта решений:

1. Если приложение работает фактически в однозадачном режиме и вас интересует только проблема предохранения экрана (без какой-либо экзотической графики), то проще всего реализовать такой режим внутри программы, без использования автономных хранителей экрана.

   Для этого создайте форму, на которую поместите элемент управления Label, содержащий какой-либо текст, а также элемент управления Timer, для которого установите свойство Interval как 1000 (то есть 1 секунда). Для формы установите свойства WindowState = Maximized, Border Style = None и выберите черный цвет в качестве фона BackColor. Теперь введите следующий код для вашей формы:

   ```
   Private Sub Form_Click()
       ' хранитель экрана выгружается, если щелкнуть форму
       Unload Me
   End Sub

   Private Sub Timer1_Timer()
       ' мигание метки каждую секунду
       Label1.Visible = Not (Label1.Visible)
   End Sub
   ```

   Соответственно в программе, когда потребуется погасить экран, нужно загрузить форму, а в необходимый момент — выгрузить ее.

   Если требуется обеспечить такой режим, чтобы по щелчку мыши хранитель исчезал на некоторое время, а потом появлялся опять, то можно написать такой код формы:

   ```
   Private Sub Form_Click()
       ' убрать хранитель
       Me.WindowState = 1 'Minimized
       Timer1.Enabled = False
   ```

```
    Timer1.Interval = 60000 ' минута
    Timer1.Enabled = True
End Sub
Private Sub Form_KeyPress(KeyAscii As Integer)
    ' если нажата Esc - выгрузить форму
    If KeyAscii = 27 Then ' Esc
        Unload Me
    End If
End Sub
Private Sub Timer1_Timer()
    If Me.WindowState = 1 Then ' хранитель свернут
        Me.WindowState = 2 ' развернуть хранитель экрана
        Timer1.Interval = 1000 ' секунда
    Else ' управляем выводом метки
        Label1.Visible = Not (Label1.Visible)
    End If
End Sub
```

2. Но что же делать, если вам действительно нужно программно сбросить внешний "фирменный" хранитель экрана? В этом случае можно воспользоваться функцией keybd_event из состава Win API, которая имитирует манипуляции с клавиатурой. (Здесь также нужно упомянуть об аналогичной функции mouse_event для имитации мыши.)

   Пример процедуры SendMyKey, которая обеспечивает имитацию записи в буфер указанного ASCI-символа, приведен на листинге 264. В этом случае для сброса внешнего хранителя экрана можно использовать такую программную конструкцию:

```
' посылка символа в буфер клавиатуры
Call SendMyKey(Chr$(9)) ' клавиша Tab
' почему-то нужно сделать вывод окна
' — тогда срабатывает посылка символа
MsgBox "Hello!"
```

   Мы не смогли понять почему, но посланный символ срабатывает только в случае, если программа потом выдает на экран еще какое-либо окно. Разумеется, если вы точно знаете, какой символ собираетесь имитировать, то можно обратиться напрямую к keybd_event, без дополнительных обращений к другим API-функциям для вычисления вспомогательных кодов. Для случая клавиши Tab это будет выглядеть следующим образом:

```
keybd_event 9, 15, 0, 0
keybd_event 9, 15, 2, 0
```

Листинг 264. Имитация нажатия клавиши клавиатуры

```
Option Explicit
Public Const KEYEVENTF_EXTENDEDKEY = &H1
Public Const KEYEVENTF_KEYUP = &H2
Declare Sub keybd_event Lib "user32" _
    (ByVal bVk As Byte, ByVal bScan As Byte, _
    ByVal dwFlags As Long, ByVal dwExtraInfo As Long)
Declare Function VkKeyScan Lib "user32" Alias "VkKeyScanA"_
    (ByVal cChar As Byte) As Integer
Declare Function CharToOem Lib "user32" Alias "CharToOemA"_
    (ByVal lpszSrc As String, ByVal lpszDst As String) As Long
Declare Function OemKeyScan Lib "user32" _
    (ByVal wOemChar As Integer) As Long
```

```
Public Sub SendMyKey(ByVal c$)
'
' Посылка одиночного ASCI-символа для имитации
' нажатия клавиши клавиатуры
    Dim vk%, scan%, oemchar$
        ' Получаем значение виртуального кода
        ' клавиши для данного символа
    vk% = VkKeyScan(Asc(c$)) And &HFF
    oemchar$ = " " ' буфер на два символа
    ' получение OEM-символа
    CharToOem Left$(c$, 1), oemchar$
        ' получение scan-кода для этой клавиши
    scan% = OemKeyScan(Asc(oemchar$)) And &HFF
        ' Нажатие клавиши
    keybd_event vk%, scan%, 0, 0
        ' Отжатие клавиши
    keybd_event vk%, scan%, KEYEVENTF_KEYUP, 0
End Sub
```

Совет 265. Как подключить VB-процедуры обратного вызова к API-функции

Оператор AddressOf, появившийся в VB версии 5.0, позволяет передавать указатель для определяемой пользователем подпрограммы, функции или свойства в API-функцию, которая затем может передавать различные данные в процедуру обратного вызова (callback procedure). Для передачи такого указателя в API-функцию следует использовать

```
AddressOf procedurename
```

в качестве аргумента в этой функции. Поэтому, например, чтобы заменить или расширить стандартную процедуру работы с сообщениями, можно написать такую подпрограмму:

```
Private Sub Form_Load()
    gWH = Me.hwnd
    OldWndProc = SetWindowLong(gWH, GWL_WNDPROC, _
    AddressOf WindowProc)
End Sub
```

где WindowProc является пользовательской процедурой. Подобная операция называется подключением обратного вызова. В этом случае аргумент WindowProc сможет перехватывать сообщения формы. В нашем случае для отключения пользовательской процедуры следует вызвать ту же самую API-функцию с указателем предыдущей процедуры приблизительно так:

```
SetWindowLong gWH, GWL_WNDPROC, AddressOf OldWndProc
```

При использовании ключевого слова AddressOf необходимо, однако, соблюдать следующие правила. Названия определяемых пользователем подпрограмм, функций или свойств, которые будут использоваться в качестве обратного вызова, должны следовать непосредственно за оператором AddressOf. Проект, в котором имеются подпрограммы, функции и свойства обратного вызова, должен содержать соответствующие описания API-функций и процедуры. Нельзя применять AddressOf ни с какими другими функциями, кроме определяемых пользователем подпрограмм, функций или свойств. Например, вы не можете использовать другую API-функцию или функцию из библиотеки типов. Обратный вызов должен находиться в стандартном модуле. И, наконец, последнее — определяемая пользователем подпрограмма или функция должна иметь тип Any либо Long.

Совет 266. Как избежать ошибок сравнения битовой маскировки при использовании API-функций в VB

При использовании API-функций достаточно часто возникает необходимость определить, содержат ли получаемые результаты определенный признак (flag) или нет. Для этого данный признак маскируется и применяется оператор And. Например, предположим, что мы хотим узнать, в каком состоянии находится текущее окно — развернуто оно или свернуто. Вначале получим установки этого окна с помощью API-функции GetWindowLong(), которую опишем в стандартном модуле следующим образом:

```
Public Const WS_MAXIMIZE = &H1000000
Public Const GWL_STYLE = -16
Public Declare Function GetWindowLong Lib _
  "user32" Alias "GetWindowLongA" _
  (ByVal hwnd As Long, ByVal nIndex As Long) As Long
```

Затем поместим на форму командную кнопку и введем такой код в ее событие Click():

```
Dim lWinStyle As Long
lWinStyle = GetWindowLong(Me.hwnd, GWL_STYLE)
If lWinStyle And WS_MAXIMIZE Then
    MsgBox "Форма развернута"
Else
    MsgBox "Форма свернута"
End If
```

Теперь запустим наш пример на выполнение и щелкнем на кнопке. Приведенный выше код определяет, содержит ли значение стиля битовое значение признака. Если да, то логическое сравнение возвращает число, которое VB интерпретирует как True (Истина). В противном случае возвращается 0 или False (Ложь).

Ошибки же могут возникнуть, если мы попытаемся проверить исключение какого-либо признака. Так, предположим, мы хотим выполнить определенные действия, когда окно находится в свернутом состоянии. Попробуем просто написать:

```
If Not (lWinStyle And WS_MAXIMIZE) Then
```

(Конечно, в данном конкретном случае мы могли бы выполнить тестирование признака WS_MINIMIZE, но иногда API-функции не имеют признака, представляющего собой противоположную установку.) К сожалению, такой условный оператор всегда возвращает значение True. Это происходит потому, что, когда мы используем оператор Not с числом, VB возвращает значение этого числа с противоположным знаком минус 1. Например, выражение Not 15 возвращает -16, которое интерпретируется VB как True. В нашем примере с битовой маскировкой выражение Not (lWinStyle And WS_MAXIMIZE) для развернутого окна имеет ненулевое значение, которое опять оценивается как True.

Чтобы избежать появления подобного "глюка", можно использовать один из двух следующих операторов:

```
If Not Cbool(lWinStyle And WS_MAXIMIZE) Then
```

или

```
If Not ((lWinStyle And WS_MAXIMIZE)<>0) Then
```

Совет 267. Используйте VB-объект RegExp для проверки синтаксиса адреса электронной почты

В современных приложениях часто бывает необходимо, чтобы пользователь вводил информацию о своей компании, включая адрес электронной почты. При этом нужно быть уверенным не только в том, что такой адрес содержит знак @ и точку, но и в том, что все остальные символы представлены буквами, числами или знаками подчеркивания. На первый взгляд такая задача может показаться не очень простой, если вы будете пользоваться только стандартными VB-функциями обработки строк. К счастью, в VB существует объект RegExp, который существенно упрощает работу.

Однако прежде чем приступить к использованию этого объекта в VB, следует загрузить библиотеку VBScript 5.0 DLL, которая находится по адресу: www.microsoft.com/msdownload/vbscript/scripting.asp

После установки библиотеки в диалоговом окне References в VB появится строка Microsoft VBScript Regular Expressions. Добавьте эту ссылку к проекту — и можно приступать к работе. Следующий код проверяет адрес электронной почты, который вводится в текстовом поле Text1:

```
Dim myReg As RegExp
Private Sub Form_Load()
    Set myReg = New RegExp
    myReg.IgnoreCase = True
    myReg.Pattern = "^[\w-\.]+@\w+\.\w+$"
End Sub
Private Sub Text1_Validate(Cancel As Boolean)
    Cancel = Not myReg.Test(Text1)
End Sub
```

Приведенный в этом коде шаблон может принимать любое количество числовых и буквенных символов, символов подчеркивания, десятичной точки и тире перед знаком @, но только числа, буквы и символы подчеркивания до и после точки.

Совет 268. Как осуществить прокрутку элемента управления ListView в VB до заданного элемента

Известно, что элемент управления ListView позволяет простым способом связать элементы списка со значками, рисунками и строками отчетов. С его помощью легко выбрать определенный элемент списка. Однако это не всегда означает, что данный элемент будет виден пользователю. Например, если список содержит много элементов, то для просмотра последнего из них необходимо пользоваться линейкой прокрутки. К счастью, элемент управления ListView предоставляет возможность программным образом перейти к заданному элементу.

Для каждого элемента списка существует метод EnsureVisible, при вызове которого указанный элемент списка выводится в видимой части элемента управления ListBox. Для иллюстрации поместите этот компонент на форму, а затем щелкните на нем правой кнопкой мыши и выберите команду Properties из быстрого меню. В диалоговом окне Properties Pages измените свойство View на 3 — lvwReport. Затем перейдите во вкладку Column Headers, выберите поле теста Insert Column и введите там "Какой элемент". Щелкните OK. И наконец напишите следующий код для события Load формы:

```
Private Sub Form_Load()
  Dim x As Integer
  With ListView1
    For x = 1 To 20
      .ListItems.Add Key:="Элемент" & x, Text:="Элемент" & x
    Next x
    .SelectedItem = .ListItems("Элемент11")
    .SelectedItem.EnsureVisible
  End With
End Sub
```

Запустите проект на выполнение. VB откроет форму, выведет окно списка и выделит в нем одиннадцатый элемент.

Совет 269. Работа в VB с формулами, представленными в виде строковых переменных

Если до того, как начать программировать на VB, вы уже работали с Microsoft Access, то вы, вероятно, ощутили нехватку в VB очень удобного метода Eval, который вычисляет заданную строковую переменную, как если бы она была кодом. Так, в Access вы могли легко вычислять математические выражения, передаваемые как строки, например выражения подобного вида:

```
iResult = Eval("(2 * 3) + 5)")
```

где iResult принимает значение 11.

Чтобы получить те же возможности в VB, вам, несомненно, пришлось обратиться к сложным функциям синтаксического разбора строк. Или же вы были вынуждены добавить к своему проекту библиотеку из Access, что существенно усложняет его для реализации столь простого метода.

Теперь, с выходом VB6, вам больше не придется прибегать к подобным ухищрениям, чтобы воспользоваться методом Eval, поскольку Microsoft включила его в элемент управления Script. С помощью этого компонента вы можете добавлять к своим приложениям возможность написания сценариев конечным пользователем. Потому-то для реализации данной возможности в состав Script и вошел метод Eval, благодаря которому вы сможете легко вычислять математические строковые переменные.

Элемент управления Script находится по адресу http://msdn.microsoft.com/scripting/. Загрузите его и выполните следующий простой пример, где для события Click командной кнопки введен такой код:

```
Private Sub Command1_Click()
    MsgBox txtFormula & " = " & SC1.Eval(txtFormula)
End Sub
```

Здесь элемент управления SC1 вычисляет математическую формулу, вводимую в поле текста txtFormula.

Совет 270. Как сделать невидимым курсор мыши в VB-приложении

APi-функция ShowCursor предоставляет простой способ сделать невидимым курсор мыши:

```
Private Declare Function ShowCursor Lib "user32" _
    ByVal bShow As Long) As Long
```

Если вы зададите параметр bShow как 0, курсор мыши исчезнет, если как 1, — появится вновь. Только помните, что при использовании этой функции курсор просто становится невидимым, а это совсем не означает, что он отключен. Чтобы продемонстрировать, что мы имели в виду, добавим приведенное выше описание функции к проекту и введем следующий код для формы:

```
Dim blnShow As Boolean
Private Sub Form_Click()
    blnShow = Not blnShow
    ShowCursor blnShow
End Sub
Private Sub Form_Load()
    blnShow = True
End Sub
Private Sub Form_QueryUnload _
    (Cancel As Integer, UnloadMode As Integer)
    ShowCursor True
End Sub
```

Запустим проект на выполнение. Щелкнем форму, и VB уберет с экрана курсор мыши. Снова щелкнем форму, и курсор появится вновь. Если бы курсор мыши был действительно отключен, то мы бы не смогли вернуть его на экран одним только щелчком формы. Теперь опять щелкнем форму, чтобы курсор сделался невидимым, и перетащим его к линейке меню интегрированной среды разработки VB. Когда невидимый курсор будет перемещаться от одной кнопки меню к другой, последняя будет приподниматься над поверхностью, сообщая о том, что она готова для щелчка. И действительно, вы по-прежнему можете пользоваться мышью для вызова команд меню или выбора объектов на экране.

Андрей Колесов, Ольга Павлова

© Андрей Колесов, Ольга Павлова, 2000
Авторский вариант. Статья была опубликована с незначительной литературной правкой в журнале "КомпьютерПресс" N 6/2000, компакт-диск.

Новость: создан Web-архив повторно используемого кода

В составе MS Office 2000 версии Developer была впервые включена полезная утилита VBA Code Librarian, обеспечивающая работу с иерархическими библиотеками исходных кодов программ. В составе этого продукта поставляется довольно большая база данных CodeLib.mdb, которую разработчик может дополнять собственными фрагментами программного кода. Отметим, что содержимое этой базы данных представляет интерес также для VB-программистов, а сама утилита может применяться для поддержки наборов кода пользователями других инструментальных средств.

С конца марта нынешнего года программисты получили возможность доступа к новому Web-сервису Code Librarian Update, с помощью которого они могут пополнять свой локальный архив повторно используемого кода.

Первое ежемесячное обновление, содержащее более 100 функций, было подготовлено Visio, подразделением Microsoft (ранее независимая компания Visio, известный разработчик ПО для проектирования и графического изображения данных, была приобретена корпорацией Microsoft в сентябре прошлого года). Этот набор программ можно загрузить с Web-страницы msdn.microsoft.com/officedev/downloads/codelibrarian.asp[41]

Выпуски Code Librarian Update будут выходить ежемесячно. Они будут содержать код, предоставленный различными сторонними компаниями (WRQ, Lawson, Elsinore Technologies и другими), которые разрабатывают продукты с поддержкой VBA. В другие продукты Microsoft, например в Microsoft MapPoint 2001 (новый продукт обработки картографических данных), также будет включен код, повышающий эффективность труда разработчиков.

Российское отделение Microsoft также довольно активно пополняет свои информационные ресурсы по VB и Office/VBA. Загляните на Web-серверы по адресам: www.microsoft.ru/msdn/library[42] и www.microsoft.ru/offext/officedev[43].

Нам пишут, мы отвечаем

Нам пришло такое письмо:

> *Просматривая CD-ROM ко второму номеру КП за 2000 год, в разделе "Советы тем, кто программирует на VB&VBA"; с удивлением прочитал Совет 244 "Как преобразовать число в строку с фиксированным количеством цифр"[44]. Неужели редакция не знает о чудодейственной функции Format? С ее помощью тот же пример занимает всего одну строку кода и без всяких дополнительных художеств:*
>
> *Format(12345, "0000000").*
>
> *Альберт*

Мы написали ответ, который считаем нужным опубликовать:

Уважаемый Альберт!

Спасибо, что читаете наши статьи, вдвойне — что читаете внимательно, втройне — что реагируете на наши недочеты. Был бы благодарен еще больше, если бы Вы (и все читатели) не только критиковали, но и делились собственным опытом работы. Если говорить об ошибках в статьях в общем случае, то их, к сожалению, полностью избежать нельзя. Мы сами их порой обнаруживаем уже в опубликованном варианте. Иногда они появляются на этапе верстки, чаще — из-за наших собственных ошибок в исходном тексте.

Мы стараемся "минимизировать" ущерб для наших читателей, сообщая о принципиальных ошибках на нашем сервере ((www.visual.2000.ru/develop/vb/[45]) и на

[41] http://msdn.microsoft.com/officedev/downloads/codelibrarian.asp
[42] http://www.microsoft.ru/msdn/library
[43] http://www.microsoft.ru/offext/officedev
[44] http://www.visual.2000.ru/develop/ms-vb/tips/0002-1.htm
[45] http://www.visual.2000.ru/develop/vb

страницах журнала. Но, честно говоря, больше всего беспокоит довольно вялая реакция читателей на эти ошибки. В результате у авторов появляются сомнения: а читает ли вообще кто-нибудь твои статьи? Использует ли кто-либо сведения, приведенные в них? Так что наша благодарность за Ваше замечание — вполне искренняя.

Что касается данного конкретного замечания, то Вы затронули довольно серьезный вопрос, который хотелось бы обсудить.

1. *Давайте уточним — редакция тут вообще ни при чем. Редакция непосредственно не занимается изучением тонкостей программирования на VB, во-вторых, она доверяет "опытным, авторитетным экспертам". Так что все упреки адресуйте не редакции вообще, а конкретно автору — Андрею Колесову, то есть мне.*
2. *Честно скажу — я не знал (до Вашего письма) о такой возможности применения функции Format. Казалось бы, на этом можно было бы поставить точку (мол, виноват, исправлюсь), но на самом деле мне здесь видится важная методическая проблема.*

   А кто вообще может похвастаться, что знает VB на все 100%? И если ответ — "никто", то является ли это недостатком пользователей или самого VB? Мне кажется, что средний пользователь представляет себе возможности системы процентов на 10, не более. И поэтому порой решает свои задачи довольно сложным образом, не подозревая, что оптимальное решение лежит рядом. Собственно, само решение освещать совершенно безграничную тему "Советов для тех..."; определяется именно этим положением вещей.

   Для одного программиста некий совет является тривиальным изложением давно известных ему вещей, для другого — ценной подсказкой. И квалификация разработчиков здесь бывает не столь важна: вполне вероятно, что в следующем совете эти два человека поменяются позициями.

   Это общая проблема современного мира информации: почти у любой проблемы есть готовое решение, но часто легче решить ее "с нуля", чем найти готовый ответ. Вопрос к Вам: откуда Вы узнали, что функцию можно применять таким образом? Представьте себе, что я — начинающий программист. Как я могу узнать о "чудодейственной функции Format"? Я просмотрел справочную информацию по всем версиям VB, начиная с 3.0 и кончая 6.0. И только в одной версии я нашел достаточно подробное описание данной функции и возможностей применения второго аргумента для числа.

   Думаю, что Вы отгадаете с первого раза, о какой версии идет речь. Конечно же, во VB3! Вопрос: откуда о возможностях функции Format сможет узнать пользователь версии 6.0?

   Однако даже если бы я знал обо всех возможностях этой функции, думаю, что публикация подобного совета (в котором вместо применения готовой функции использовала несколько более сложная конструкция) все равно имела бы смысл. Вообще говоря, я считаю, что VB (и современные средства программирования вообще) явно перенасыщены дублирующими функциями. Например, зачем создавать специальную функцию проверки существования файла, если ее можно выполнить уже имеющейся функцией Dir?

3. *Давайте вернемся к упомянутому Совету 244. На самом деле приведенный Вами пример — a$ = Format (12345, "0000000") — можно реализовать еще проще: a$="0012345". Ведь в*

нашем совете речь шла о создании функции, которая преобразовывала произвольное число SourceNumber в строку с заданным числом цифр Lend. С использованием функции Format это будет выглядеть так:

4.
```
a$ = Format(SourceNumber, String$(Lend, "0"))
```

В нашем варианте это выглядит немного более замысловато:

```
If 10^Lend > SourceNumber Then
  a$ = Right$(10^Lend + SourceNumber, Lend)
Else
  a$ = Str$(SourceNumber)
End If
```

Основная сложность нашей конструкции — проверка на допустимость параметров функции. Но обратите внимание на преимущество последнего варианта: это может сделать каждый программист, на любой версии Basic. (Даже на QBasic — поверьте, что пользователей этой системы еще довольно много.)

Нам кажется, что демонстрация того, что многие функции могут быть решены неким тривиальным набором операций, очень важна, так как она стимулирует разработчика к поиску собственного решения (чтобы не ждать, когда Microsoft решит осчастливить человечество своим встроенным оператором).

5. *И все же основная цель нашей рубрики — поделиться собственным опытом, своими находками. Действительно, порой имеет место "изобретение велосипеда". Но чем больше людей (разработчиков) будут участвовать в этом обсуждении, тем меньше будет таких "велосипедов".*

Короче говоря: спасибо за письмо. Оставайтесь с нами.

Присылайте свои советы, замечания, вопросы.

Андрей Колесов,
один из ведущих рубрики "Советы для тех..."

Глобальный совет: работайте с лицензионными копиями программ

Наш многолетний опыт общения с пользователями VB позволяет нам сделать уверенный вывод: пиратские копии программ, продаваемые в Митино, на Горбушке и пр., отличаются от оригиналов. Мы не являемся службой поддержки Microsoft (не несем каких-то обязательств перед корпорацией), но мы придерживаемся такого подхода при анализе проблем, с которыми к нам обращаются читатели. Если при воспроизведении некоего примера мы получаем различные результаты (у нас работает, а у спрашивающего — нет), то сразу задаем вопрос: какая копия VB у вас стоит? Дело в том, что проблемы довольно часто возникают именно из-за применения программ неизвестного происхождения.

Мы не хотим читать нотации и повторять тезис о том, что применение пиратских копий является нарушением российского законодательства. Но все же — если кто-то желает сэкономить деньги, то, отправляясь в Митино, он должен хотя бы понимать, что это может обернуться для него некоторыми потерями... Для примера расскажем одну историю.

В начале весны к нам в офис приехал за консультацией человек, которому мы помогали в разработке его программы лет шесть назад, во времена Basic/DOS. У него не работала программа, которую он скачал из Интернета, а она была ему очень интересна.

Проблема решалась тривиально — ошибка была в "левой" копии VB 6, установленной на ноутбуке клиента (не работало подключение OCX-компонентов). Но самое поразительное было в другом — у него стояла программа с полностью русифицированным интерфейсом (Microsoft выпускает локализованные версии VB только с русской справкой). Но наш пользователь при этом был полностью уверен, что его инструмент — точная копия оригинала.

Мы часто ищем истоки российского пиратства в низкой покупательной способности населения. Но вот что интересно — наш программист приехал к нам на иномарке с довольно приличным ноутбуком. Неужели он не понимает, что время, которое он тратит на борьбу с ошибками пиратских копий, — это деньги, которые он теряет?

Андрей Колесов, Ольга Павлова

© Андрей Колесов, Ольга Павлова, 2000
Авторский вариант. Статья была опубликована с незначительной литературной правкой в журнале "КомпьютерПресс" N 6/2000, компакт-диск.

Совет 271. Избегайте использования переменных типа Variant

Наш Совет 259 (об использовании операторов Print # и Write #)[46] был основан на замечании, которое прислал наш читатель Константин Абакумов по поводу конструкций, использованных нами в коде утилиты Summa.vbp (сумма прописью, Совет 228[47]). Кроме того, он обнаружил еще одну "дырочку" в этой программе: оказалось, что следующая конструкция у него давала неверный результат:

```
Dim InVal
...
Input #1, InVal                  ' здесь у Константина не срабатывала
If InVal = True Then Call ...    ' проверка при вводе значения True
...
```

Константин предложил заменить последнюю строку на такой работоспособный вариант:

```
If InVal Then Call ...    ' при вводе значения True
```

Совершенно очевидно, что оба варианта в принципе тождественны. Не говоря уже о том, что у нас исходный вариант надежно работал. В чем же дело?

Проблема заключалась в том, что мы работали в среде VB5 (мы сознательно используем эту версию, если нет нужды в применении новшеств VB6), а наш читатель — в VB6. Судя по всему, в VB6 имеется ошибка, в результате чего внешне правильная конструкция оказалась неработоспособной.

[46] http://www.visual.2000.ru/develop/ms-vb/tips/0004-1.htm
[47] http://www.visual.2000.ru/develop/ms-vb/tips/0001-1.htm

Казалось бы, здесь можно поставить точку: обнаружена ошибка — избегайте ее повторения. Но на самом деле за этим частным случаем видится весьма принципиальная проблема.

Мы считаем, что VB (как и многие другие современные системы программирования) обладает серьезным технологическим недостатком, который заключается в широком использовании неявного преобразования типов данных. С точки зрения классических принципов программирования конструкция типа:

```
srtVal$ = IntVal%
```

является просто недопустимой — преобразование типов должно выполняться только с помощью специальных функций в явном виде, например так:

```
SrtVal$ = Str$(IntVal%)
```

И это при том, что ключевой идеей программирования всегда была необходимость четкого контроля за типами данных. Говоря упрощенно, проблема заключается в том, что при смешении типов переменных часто возникает двусмысленность в порядке преобразования данных и поэтому может получаться довольно неожиданный вариант.

Строго говоря, неявное преобразование данных недопустимо даже в арифметических операциях (из целых в вещественные и наоборот). Например, результат такого выражения:

```
sVal! = 1/3*6
```

в зависимости от желания разработчика компилятора может быть равным и 0 и 2. И в обоих случаях разработчик будет прав, так как нет жестких правил порядка преобразования данных.

Что же касается конструкции (вполне допустимой в VB) типа

```
strTrue$ = "True"
If strTrue$=True Then ...,
```

то она вообще представляется кошмарной.

Логическим выводом из вышесказанного является то, что появление некоторого универсального типа данных, которым в VB является Variant, является грубым нарушением классических принципов программирования. Желание помочь программистам ("вам не нужно думать о форме представления данных") на самом деле усложнило разработку программ.

Пример тому — приведенная выше ситуация, когда VB сам запутался в том, как нужно интерпретировать переменную InVar. Кто-то отнесет это к невнимательности разработчиков Microsoft, но на самом деле появление этой ошибки было предопределено стратегией на смешение типов данных.

Каемся: мы были не правы, что использовали в утилите Summa общую переменную InVar (по умолчанию — Variart) для обработки ввода нескольких параметров разных типов данных. Мы исправили код, который теперь выглядит так:

```
Dim InBool As Boolean, InText$, InVal%
```

```
  Input #1, InBool: If InBool Then Call LabelSet(1)
  Input #1, InText: txtW1.Text = InText
```

ИЗБЕГАЙТЕ *использования переменных типа Variant! Применяйте фиксированные типы переменных и функции явного преобразования типов данных. Если у вас есть доводы против этого тезиса — присылайте нам*[48] *свои соображения.*

Совет 272. Вывод анимационных GIF-файлов в VB

Хотя элемент управления Picture позволяет отображать на экране графические изображения, он выводит только первый рисунок из анимационного GIF-файла. Чтобы полностью показать такой файл, не создавая заново всю последовательность изображений, можно воспользоваться элементом управления WebBrowser (который, правда, недоступен на компьютерах, где отсутствует Internet Explorer 3.0 или выше). Для этого установите компонент Microsoft Internet Controls (команда Project|Components), и тогда на панели инструментов Visual Basic появится элемент управления WebBrowser. Поместите его на форму, а затем в событии Form_Load() введите следующий код:

```
WebBrowser1.Navigate "filespec"
```

где filespec — полное имя GIF-файла (включая путь к нему) или URL-адрес. Запустите программу на выполнение, и Visual Basic выведет указанный файл.

К сожалению, WebBrowser одновременно выведет вертикальную линейку прокрутки на правой стороне формы, что не вполне сочетается с графическим изображением. Чтобы отключить ее, как ни удивительно, можно применить тот же самый метод, что обычно реализуется в HTML-коде:

```
WebBrowser1.Navigate @about:<html>" & _
    "<body scroll='no'><img src='filespec'>" & _
    </img></body></html>"
```

Теперь, когда вы запустите программу, Visual Basic выведет графическое изображение без линейки прокрутки.

Совет 273. Как прочитать значения Системного Реестра в VB без помощи функций API

В некоторых случаях бывает необходимо управлять Реестром Windows в VB, не прибегая к использованию функций API. Для этого существует библиотека Registry Access Functions (RegObj.dll), позволяющая создавать объект Registry, при помощи которого можно управлять конкретными ключами. Это может пригодиться, например, в следующем случае.

В предыдущем Совете мы продемонстрировали использование элемента управления WebBrowser. Однако, как мы уже отмечали, WebBrowser доступен только в том случае, если на машине-приемнике также установлен Internet Explorer. Поэтому, прежде чем приступить к выводу анимации, было бы целесообразно определить, имеется там IE или нет. Для этого вначале установите в своем проекте ссылку к библиотеке Registry Access Functions (команда Project|References) и введите код, аналогичный тому, что приводится здесь:

[48] mailto:akolesov@online.ru?subject=VariantType

```
Dim myReg As New Registry, KeyFound As Boolean
Dim HasIE As Boolean, sValue As String
    sValue = ""
    HasIE = False
    KeyFound = myReg.GetKeyValue(HKEY_LOCAL_MACHINE, _
        "Software\Microsoft\Windows\CurrentVersion\" & _
        "App Paths\IEXPLORE.EXE", "Path", sValue)
    If KeyFound Then HasIE = (sValue <> "")
     'выводит полное название каталога,
     'где установлен IE
    If HasIE Then MsgBox sValue
    Set myReg = Nothing
```

Совет 274. Реализация проверки правописания в RTF-документах в VB

Для проверки правописания в RTF-документах можно было бы прибегнуть к следующему способу: внедрить Word в свое приложение (этот вариант приведен ниже, в Совете 282). Однако мы рекомендуем воспользоваться элементом управления WebBrowser, о котором шла речь в двух предыдущих Советах и с помощью которого можно организовать такую проверку. Для этого вам понадобится метод ExecWB() с флагом OLECMDID_SPELL. В качестве иллюстрации поместите WebBrowser на форму и введите такой код в событие Form_Load():

```
WebBrowser1.Navigate "filespec"
```

Замените filespec любой строкой, которая указывает путь к RTF-файлу. Затем добавьте к форме командную кнопку, для которой в событии Click напишите следующий код:

```
WebBrowser1.ExecWB OLECMDID_SPELL, OLECMDEXECOPT_DODEFAULT
```

Запустите проект на выполнение. Элемент управления WebBrowser выведет указанный RTF-документ, и, когда вы щелкнете командную кнопку, VB активизирует установленную на вашем компьютере программу проверки правописания.

Совет 275. Как реализовать ссылки к набору записей объекта Command, содержащегося в конструкторе Data Environment

Каждый объект Command в новом конструкторе Data Environment в VB 6.0 имеет соответствующий ему набор записей. Для того чтобы в коде программы сослаться на этот объект, необходимо добавить к его имени префикс rs. Так, если конструктор Data Environment содержит объект Command с именем cmdSomeQuery, то для ссылки к набору записей этого объекта напишите примерно такую строку кода:

```
Set rst = DataEnvironment1.rscmdSomeQuery
```

Совет 276. Сжимайте длинные имена файлов с помощью библиотеки SHLWAPI

Использование функции PathCompactPath из библиотеки SHLWAPI — один из способов сжатия длинных имен файлов, благодаря которому часть имени слева заменяется на многоточие (...). Для работы с данной API-функцией требуется описать ее следующим образом:

```
Private Declare Function _
```

```
PathCompactPath Lib "shlwapi"_
    Alias "PathCompactPathA" _
    (ByVal hDC As Long, ByVal _
    lpszPath As String, _
    ByVal dx As Long) As Long
```

Как видно из этого описания, функция PathCompactPath содержит три аргумента: первый из них задает дескриптор контекста устройства (device context handle), второй — имя файла, которое будет сжиматься, и, наконец, последний — ширину в пикселах того компонента на форме, куда мы хотим поместить это имя. Так, чтобы вывести компактное имя файла в метке с именем lblEllipsis, напишем следующий код для события Click() командной кнопки:

```
Private Sub Command1_Click()
Dim lhDC As Long, lCtlWidth As Long
Dim FileSpec As String
    FileSpec = "C:\MyFolder\VisualBasic\" & _
        "MyReallyWayTooLongFolderName\ButWhoCares\IhaveTheAPI.doc"
    Me.ScaleMode = vbPixels
    lCtlWidth = lblEllipsis.Width - Me.DrawWidth
    lhDC = Me.hDC
    PathCompactPath lhDC, FileSpec, lCtlWidth
    lblEllipsis.Caption = FileSpec
End Sub
```

Совет 277. Еще один способ чтения нескольких элементов списка

Обычное правило чтения выделенных элементов окна списка, поддерживающего выбор нескольких позиций (иными словами, того, у которого свойство MultiSelect отлично от 0), гласит, что необходимо организовать в цикле просмотр всех элементов и проверить у них значение свойства Selected. Однако такой подход (как и при работе с любыми циклами) может значительно снизить быстродействие создаваемого вами приложения, особенно на менее скоростных процессорах. В качестве более быстрой и элегантной альтернативыпредлагаем вам воспользоваться API-функцией SendMessage().

Как известно, с помощью данной функции можно отправить сообщение в одно или несколько окон. Ее описание выглядит следующим образом:

```
Private Declare Function SendMessage Lib "user32" _
   Alias "SendMessageA" (ByVal hwnd As Long, ByVal wMsg _
    As Long, ByVal wParam As Long, lParam As Any) As Long
```

Поскольку мы хотим прочитать все выделенные элементы списка, необходимо отправить константу LB_GETSELITEMS в качестве аргумента wMsg и описать ее так:

```
Private Const LB_GETSELITEMS = &H191
```

По существу сообщение LB_GETSELITEMS заполняет массив значениями индексов всех выделенных элементов. В результате вместе с функцией SendMessage() необходимо передать два дополнительных аргумента. Первый из них должен содержать максимальное количество выделенных элементов, получить которое можно с помощью свойства SelCount элемента управления ListBox. Второй же аргумент должен хранить переменную массива, которую мы хотим заполнить значениями индексов.

Следующий пример демонстрирует, как можно использовать функцию SendMessage(). Поместите на форму элемент управления ListBox1, установите его свойство MultiSelect как 1-Simple и заполните список. Затем добавьте командную кнопку Command1 и напишите следующий код для ее события Click:

```
Dim ItemIndexes() As Long, x As Integer, iNumItems As Integer
NumItems = ListBox1.SelCount
If iNumItems Then
    ReDim ItemIndexes(iNumItems - 1)
    SendMessage ListBox1.hwnd, LB_GETSELITEMS, iNumItems, _
    ItemIndexes(0)
End If
For x = 0 To iNumItems - 1
    MsgBox ListBox1.List(ItemIndexes(x))
Next x
```

Обратите внимание, что переменная iNumItems, после того как она передается в функцию SendMessagen, хранит общее число выделенных элементов списка, а массив ItemIndexes — значения индексов выделенных элементов. При этом вы должны передать указатель к массиву ItemIndexes, а не сам массив, то есть в функцию SendMessage передается ItemIndexes(0), а не ItemIndexes().

Совет 278. Как можно просто связать элементы двух списков

Предположим, что на вашей форме есть два окна списка, связанных таким образом, что элементы второго списка зависят от того, какой элемент выбран в первом списке. Если вы будете решать подобную задачу с помощью стандартных массивов, то вам придется написать огромное количество кода. Мы же предлагаем использовать функции работы с массивами, что должно почти вдвое сократить объем кода. Кроме того, применение этих функций значительно упрощает добавление элементов списка, поскольку отпадает необходимость менять размерности массивов, а это, в свою очередь, снижает вероятность появления ошибок. Например, чтобы добавить элемент "Конфеты" к первому списку, просто вставьте в основной массив подмассив следующего вида:

```
Array(3, "Конфеты", "Мишка на Севере", "Белочка", "Красная Шапочка")
```

В предлагаемом нами коде каждый элемент первого списка имеет один подмассив, а элементы каждого подмассива определяются следующим образом: сначала пишется количество элементов второго списка, затем имя элемента первого списка, для которого и создается данный подмассив, далее идут элементы второго списка.

Чтобы протестировать приведенный ниже код, создайте проект Standard EXE, а потом поместите на форму два окна списка, оставив без изменения их имена List1 и List2:

```
Private varArray As Variant
Private varSubArray As Variant
Private Sub Form_Initialize()
    varArray = Array(Array(4, "Фрукты", 'Яблоки", _
    "Апельсины", "Персики", "Груши", Array(5, _
    "Овощи", "Горох", "Бобы", "Кукуруза", "Свекла", _
    "Лук"), Array(3, "Ежедневные продукты", _
    "Молоко", "Сметана", "Масло"))
End Sub
Private Sub Form_Load()
  Dim intIndex1 As Integer
  With List1
```

```
    For intIndex1 = 0 To UBound(varArray)
      varSubArray = varArray(intIndex1)
      .AddItem varSubArray(1)
    Next intIndex1
   .ListIndex = 0
  End With
End Sub

Private Sub List1_Click()
  Dim intIndex2 As Integer
  With List2
    varSubArray = varArray(List1.ListIndex)
    .Clear
    For intIndex2 = 0 To varSubArray(0) - 1
      .AddItem varSubArray(intIndex2 + 2)
    Next intIndex2
    .ListIndex = 0
    .Refresh
  End With
End Sub
```

Совет 279. Как реализовать работу с несколькими дисководами CD-ROM с помощью MCI-интерфейса

Интерфейс MCI (Media Control Interface — Интерфейс управления средой) позволяет поддерживать несколько устройств для чтения аудиокомпакт-дисков. Для этого следует просто указать имя дисковода в команде MCI Open. Вначале поместите окно списка List1 на форму, а затем, чтобы определить, какие дисководы предназначены для чтения компакт-дисков, напишите в разделе General Declarations формы такой код:

```
Private Declare Function GetDriveType Lib _
    "kernel32" Alias "GetDriveTypeA" (ByVal _
    nDrive As String) As Long
Private Declare Function mciSendString Lib _
    "winmm.dll" Alias "mciSendStringA" (ByVal _
    lpstrCommand As String, ByVal lpstrReturnString _
    As String, ByVal uReturnLength As Long, ByVal _
    hWndCallback As Long) As Long
Private Const DRIVE_CDROM = 5
```

Подпрограмма Form_Load заполняет окно списка обнаруженными на компьютере именами дисководов CD-ROM:

```
Sub Form_Load()
  Dim k As Long
  For k = Asc("A") To Asc("Z")
    If GetDriveType(Chr$(k) & ":") = DRIVE_CDROM Then
      List1.AddItem Chr$(k) & ":"
    End If
  Next
End Sub
```

А чтобы действительно открыть дверцу дисковода, куда можно будет вставить компакт-диск, введите такой код в событие List1_dblClick:

```
Private Sub List1_DblClick()
  mciSendString "open"; & _
    List1.List(List1.ListIndex) & _
    "type cdaudio alias cdaudi", _
```

```
        vbNullString, 0, 0
    mciSendString "set cdaudio door open", _
        vbNullString, 0, 0
    mciSendString "close cdaudio", _
        vbNullString, 0, 0
End Sub
```

Совет 280. Как обрабатывать события, связанные с изменением свойств шрифта

Предположим, вы хотите выполнить какое-нибудь действие при изменении любых свойств шрифта конкретного элемента управления или формы. Для этого вам понадобится ключевое слово WithEvents. (Не забудьте установить ссылку OLE Automation в диалоговом окне References с помощью команды Project|References):

```
Private WithEvents fntAny As StdFont
```

Затем напишите подпрограмму, которая будет реагировать на изменения свойств шрифта:

```
Private Sub fntAny_FontChanged(ByVal _
    PropertyName As String)
    Select Case PropertyName
        Case "Name"
            'Какое-либо действие
        Case "Size"
            'Какое-либо действие
        Case "Italic"
            'Какое-либо действие
        Case "Bold"
            'Какое-либо действие
        Case "Underline"
            'Какое-либо действие
        ' ...
        ' Подобным образом можно расширить
        ' функциональные возможности для
        ' каждого свойства шрифта
    End Select
End Sub
```

Теперь осталось только установить свойство Font формы или элемента управления как fntAny. Так, если вы хотите отслеживать изменения свойств шрифта формы, напишите следующий код в событии Form_Load:

```
Set fntAny = Me.Font
' Если вы имеете дело с элементом управления,
' тогда Control.Font
```

Совет 281. Как определить, какой именно объект не может быть создан

При разработке крупного VB-приложения, использующего сотни COM-объектов, вы можете в какой-то момент получить сообщение об ошибке такого вида: "429 can't create object". К сожалению, эта информация не очень-то поможет вам определить, какой именно объект не может быть создан. Чтобы преодолеть подобное ограничение, можно написать функцию на базе функции CreateObject из библиотеки Visual Basic runtime objects and procedures:

```
Public Function CreateObject(sProgID As String) As Object
```

```
    '
    On Error GoTo CreateErr
    ' Вызов функции CreateObject из библиотеки
    ' VB runtime objects and procedures
    Set CreateObject = VBA.CreateObject(sProgID)
    Exit Function

CreateErr:
    ' возвращает сообщение об ошибке вместе
    ' с именем объекта, который не может
    ' быть создан
    Err.Raise Err.Number, "CreateObject Wrapper", _
    Err.Description & ": '" & sProgID & "'"
End Function
```

Совет 282. Используйте возможности Office для проверки правописания в элементе управления RichTextBox

VB позволяет осуществить интеграцию возможностей проверки правописания, имеющихся в Microsoft Word 97/2000, в VB-приложения, сохраняя при этом параметры форматирования внутри элемента управления RichTextBox. Продемонстрируем это на таком примере:

1. Создайте проект Standard EXE в VB.
2. Добавьте к панели инструментов элемент управления RichTextBox (команда Project|Components).
3. Добавьте ссылку к библиотеке Microsoft Word 8.0 Object Library (для Word 2000 — Microsoft Word 9.0 Object Library).
4. Разместите на форме элементы управления RichTextBox и CommandButton.
5. Переименуйте компонент RichTextBox в rtfText.
6. Установите свойство Caption командной кнопки как "Правописание".
7. Введите в событие Click командной кнопки следующий код:

```
8.   Private Sub Command1_Click()
9.     On Error GoTo SpellCheckErr
10.    Dim oWord As Object
11.
12.    Set oWord = CreateObject("Word.Application")
13.    '
14.    ' сохраняет содержимое компонента RichTextBox
15.    'во временном файле
16.    rtfText.SaveFile "c:\Vb-db\Test.RTF"
17.    '
18.    ' открывает сохраненный файл и проводит
19.    ' в нем проверку правописания
20.    oWord.Documents.Open ("c:\Vb-db\Test.RTF")
21.    oWord.ActiveDocument.SpellingChecked = False
22.    oWord.Options.IgnoreUppercase = False
23.    oWord.ActiveDocument.CheckSpelling
24.    '
25.    ' сохраняет изменения в RTF-файле
26.    ' и закрывает его
27.    oWord.ActiveDocument.Save
28.    oWord.ActiveDocument.Close
29.    oWord.Quit
30.    '
31.    ' загружает изменения в элемент
32.    ' управления RichTextBox
33.    rtfText.LoadFile "c:\Vb-db\Test.RTF"
34.
```

```
35.    Set oWord = Nothing
36.    Screen.MousePointer = vbDefault
37.    MsgBox "Проверка правописания закончена", _
38.            vbInformation, "Правописание"
39.    Exit Sub
40.    '
41.  SpellCheckErr:
42.     MsgBox Err.Description, vbCritical, "Правописание"
43.     Set oWord = Nothing
44.  End Sub
```

45. Сохраните проект и запустите его на выполнение.
46. Введите какой-нибудь текст в элемент управления RichTextBox и затем щелкните кнопку "Правописание". Вы попадете в среду Word, где и осуществите проверку правописания.

Совет 283. Используйте библиотеку ADOX, чтобы установить существование внутренних объектов баз данных

Вскоре после выпуска библиотеки ADO компания Microsoft создала для нее расширение, называемое ActiveX Data Objects Extensions, или просто ADOX, и включающее большую часть функций DAO, которые не вошли в состав стандартной версии ADO. С помощью библиотеки ADOX можно легко определить, входит ли конкретная таблица, разрез данных или запрос в базу данных.

Для этого установите ссылку к библиотеке Microsoft ADO Ext. for DDL and Security. В упрощенном виде ADOX состоит из объекта Catalog, который включает коллекции объектов, описывающих базу данных и содержащиеся в ней таблицы и разрезы данных. Чтобы проверить наличие конкретной таблицы в коллекции Tables, можно пройти шаг за шагом по всей коллекции и сравнить имя каждого элемента с заданной строкой либо воспользоваться предлагаемым ниже способом:

```
Private Sub Command1_Click()
    Dim con As New ADODB.Connection
    Dim cat As New ADOX.Catalog
    Dim tbl As ADOX.Table
    Dim tblName As String
    con.Open "Provider=Microsoft.Jet.OLEDB.4.0;" & _
        "Data Source=c:\Vb-db\Biblio.mdb;"
    Set cat.ActiveConnection = con
    On Error Resume Next
    tblName = "Titles"
    Set tbl = cat.Tables(tblName)
    If tbl Is Nothing Then
        MsgBox "Таблица " & tblName & " не существует"
    Else
        MsgBox "Таблица " & tblName & " существует"
        Set tbl = Nothing
    End If
    Set cat = Nothing
    Set con = Nothing
End Sub
```

Совет 284. Используйте оператор SendKeys

В нашем Совете 264 мы рассказывали о том, как программным образом имитировать нажатие клавиши с помощью обращения к соответствующей API-функции. Оказывается (каемся — не знали), что в VB можно использовать для этих целей встроенный оператор SendKeys. Например, посылка кода Enter выглядит следующим образом:

```
SendKeys "{Enter}"
```

Такая программная имитация нажатия клавиши может быть полезна при создании макрокоманд в MS Office. Ведь некоторые команды в ходе их выполнения выдают диагностические сообщения с требованием подтвердить выполнение операции или выбрать нужный вариант ее реализации. Использование SendKeys позволяет автоматически выполнить подобные "нажатия" кнопок.

Андрей Колесов, Ольга Павлова

© Андрей Колесов, Ольга Павлова, 2000
Авторский вариант. Статья была опубликована с незначительной литературной правкой в журнале "КомпьютерПресс" N 7/2000, компакт-диск.

Совет 285. Как получить значения цветовой палитры

В состав VB входит удобное средство преобразования отдельных значений Red, Green и Blue в одно цветовое значение типа Long — это функция RGB. К сожалению, VB не позволяет проводить обратное преобразование, но вы можете получить конкретные цветовые значения из шестнадцатеричного представления значения типа Long, создаваемого функцией RGB. Для этого создадим следующую функцию и поместим ее в стандартном модуле программы:

```
Public Type RGB_Type
  R As Long
  G As Long
  B As Long
End Type

Public Function ToRGB(ByVal Color _
  As Long) As RGB_Type
  '
  Dim ColorStr As String
  ColorStr = Right$("000000" & Hex$(Color), 6)
  With ToRGB
    .R = Val("&h" & Right$(ColorStr, 2))
    .G = Val("&h" & Mid$(ColorStr, 3, 2))
    .B = Val("&h" & Left$(ColorStr, 2))
  End With
End Function
```

Чтобы воспользоваться данной функцией, поместите в форму какое-либо изображение, задав необходимое имя в свойстве Picture формы, а затем введите такой код:

```
Private Sub Form_MouseUp(Button _
  As Integer, Shift As Integer, _
  X As Single, Y As Single)
  '
  Dim RGB_Point As RGB_Type
  RGB_Point = ToRGB(Point(X, Y))
  With RGB_Point
    Me.Caption = "R = " & .R & " G = " & .G & " B = " & .B
  End With
End Sub
```

Запустите программу на выполнение. Щелкая мышью на различных частях изображения, вы будете видеть в заголовке формы соответствующие значения RGB. Обратите внимание, что при работе в VB3 вам нужно получать эти значения по отдельности, поскольку VB не поддерживал возвращение заданных пользователем типов данных до версии VB4. Подобное преобразование можно осуществить более быстрым способом, если воспользоваться командой LSet, которая копирует содержимое одного заданного пользователем типа данных (user-defined type, udt) в другой. Для этого заменим функцию, находящуюся в стандартном модуле программы, на такую:

```
Public Type RGB_Type
     R As Byte
     G As Byte
     B As Byte
     Filler As Byte
End Type
Private Type RGB_Full_Type
     lngRGB As Long
End Type

Public Function ToRGB(ByVal _
  vlngColor As Long) As RGB_Type
  '
  Dim udtRGBFull As RGB_Full_Type
  udtRGBFull.lngRGB = vlngColor
  LSet ToRGB = udtRGBFull
End Function
```

Совет 286. Будьте внимательны при использовании оператора Debug.Print

Вопреки распространенному мнению, операторы Debug.Print не всегда удаляются из исполняемых файлов. Продемонстрируем, например, такой случай. Создайте новый проект, поместите на форму командную кнопку Command1 и напишите для нее такой код:

```
Private Sub Command1_Click()
  Debug.Print DebugTime
End Sub
Public Function DebugTime()
  MsgBox "Привет!"
End Function
```

Скомпилируйте программу, запустите ее на выполнение и щелкните командную кнопку. На экране, как ни удивительно, появится окно сообщения. Это, конечно же, искусственная ситуация, но можно легко представить себе случаи, когда оператор Debug.Print используется для печати возвращаемого значения функции. Если переменные передаются как параметры ByRef и если функция изменяет значения этих переменных, то подобная ошибка распространится и на исполняемый файл, а обнаружить ее будет крайне трудно.

Таким образом, сформулируем основные выводы:

- Передавайте параметры с помощью ключевого слова ByVal, за исключением тех случаев, когда вы уверены, что не будете изменять их, или когда вы применяете их в качестве "выходных" параметров.
- Соблюдайте осторожность при использовании оператора Debug.Print. Он может выполнять больше действий, чем вы думаете.

Совет 287. Как упростить математические вычисления в VBA

Существует возможность упростить некоторые вычислительные процедуры в Excel, а также добавить в Word отсутствующие в нем функции. К сожалению, большинство учебных пособий по VBA в Office 97 предназначены для изучения нематематических функциональных возможностей пакетов, например форматирования текста или вывода графических изображений.

Отсутствие необходимой документации ставит в затруднительное положение тех, кто хочет использовать макросы для решения своих математических задач, поскольку обработка числовых значений в ячейке таблицы Word происходит иначе, чем в ячейке электронной таблицы Excel. Потратив некоторое время, вы, конечно же, найдете необходимое описание синтаксиса математических функций в справочной системе Word или Excel. Кроме того, можно воспользоваться руководством Microsoft Office 97: Visual Basic Programmer's Guide, изданным корпорацией Microsoft (русская версия была выпущена "Русской Редакцией"), где рассматриваются практически все вопросы на данную тему.

Здесь мы приводим два примера, которые в явном виде демонстрируют разницу при работе в Word и Excel. Они вычисляют кубическую сумму значений первых восьми ячеек столбца 1 и помещают результат в девятую ячейку столбца 1:

- Для Word VBA:

```
Sub ColumnMath()
  Dim x As Long, i As Long
  Dim myTable As Table
  Dim myStr As String
  x = 0
  Set myTable = ActiveDocument.Tables(1)
  For i = 1 To 8
       x = x + myTable.Cell(i, 1).Range.Calculate ^ 3
  Next I
  myStr = Str(x)
  myTable.Cell(9, 1).Range. InsertAfter (myStr)
End Sub
```

- Для Excel VBA:

```
Sub ColumnMath()
  Dim x As Long, i As Long
  Sheets("Sheet1").Activate x = 0
  For i = 1 To 8
    x = x + Cells(i, 1).Value ^ 3
  Next i
  Range("a9").Value = x
End Sub
```

Данные вычисления могут быть проведены целиком в рамках функциональных возможностей Excel, однако для этого потребуется создать дополнительный столбец. А вот как выполнить подобные действия внутри Word? Попытаемся разобраться в этом. Напомним, что Word VBA имеет неинтуитивный синтаксис для обработки ячеек. Чтобы прочитать значение ячейки, следует использовать ключевое слово Calculate.

А чтобы записать какую-либо величину в ячейку, надо вначале преобразовать ее в строковую переменную, а затем применить абсолютно неочевидное ключевое слово

InsertAfter. Нумерация таблиц в документе Word осуществляется последовательно, поэтому в первом примере мы имеем дело с первой таблицей документа. Каждая таблица рабочей книги Excel представляет собой одну большую таблицу, поэтому во втором примере мы работаем с первой таблицей рабочей книги. Кроме того, во втором примере Range("a9") можно заменить на Cells(9, 1) или на один из других вариантов, требующих использования ключевого слова Range.

Совет 288. Как вычислить интервал между двумя датами, измеряемый в разных единицах

Как вы уже наверняка знаете, для вычисления интервала между двумя датами можно использовать встроенную в VB функцию DateDiff. Но работать с этой функцией нужно очень внимательно, с учетом входящих в нее ограничений.

Отгадайте такую загадку. Заданы две даты в виде переменных DateStart и DateFinish. Чтобы определить временной интервал между ними, мы написали такую процедуру:

```
Print "Интервал в годах = "; DateDiff("yyyy", DateStart, DateFinish)
Print "Интервал в месяцах= "; DateDiff("m", DateStart, DateFinish)
Print "Интервал в днях = "; DateDiff("d", DateStart, DateFinish)
Print "Интервал в часах = "; DateDiff("h", DateStart, DateFinish)
Print "Интервал в минутах = "; DateDiff("n", DateStart, DateFinish)
Print "Интервал в секундах= "; DateDiff("s", DateStart, DateFinish)
```

И получили такой парадоксальный результат:

```
Интервал в годах = 1
Интервал в месяцах = 1
Интервал в днях = 1
Интервал в часах = 1
Интервал в минутах = 1
Интервал в секундах = 1
```

ВОПРОС. *Почему так произошло и о каких датах шла речь?*

ОТВЕТ. Дело в том, что функция DateDiff определяет временной интервал элементарно — в соответствии с заданным первым параметром просто отбрасывает значения даты "после этой точки". То есть если вы задали "день", то отбрасываются часы (0 часов), если месяц — дни (первое число месяца). В соответствии с этим алгоритмом получается, что между 31 мая 2000-го и 1 июня 2000-го в единицах "месяц" разница — один месяц (что в определенном смысле совершенно верно).

В нашем же примере исходные значения даты были равны

```
DateS = "31.12.2000 23:59:59"
DateF = "01.01.2001"
```

Изменение показателя текущего момента на одну секунду привело к изменению минут, часов, суток, месяца и года (и даже века и тысячелетия). Очевидно, что самое точное определение интервала дается в данном случае в секундах (этой точности вполне достаточно для решения большинства бытовых и деловых проблем). Но как интерпретировать величину типа 12 345 678 сек? Конечно, желательно получить информацию в более привычных единицах — месяцах, днях, минутах. В таких случаях

вам поможет подпрограмма DateIntervals, позволяющая передавать две даты и свои собственные переменные для указанного интервала:

```
Public Sub DateIntervals(ByVal DateS _
  As Date, ByVal DateF As Date, ParamArray Prams())
  If UBound(Prams) < 0 Then Exit Sub

  Dim i As Long, itr As String
  '
  ' Если не задан день, то считаем его "сегодняшним"
  If DateValue(DateS) = 0 Then DateS = DateS + DateValue(Now)
  If DateValue(DateF) = 0 Then DateF = DateF + DateValue(Now)
  '
  For i = 0 To IIf(UBound(Prams) > 5, 5, UBound(Prams))
    If Not IsMissing(Prams(i)) Then
      If i = 0 Then
        itr = "yyyy"
      Else
        itr = Mid$("mdhns", i, 1)
      End If
      Prams(i) = DateDiff(itr, DateS, DateF)
      If DateAdd(itr, Prams(i), DateS) <= DateF Then _
        Prams(i) = Prams(i) - 1
      DateS = DateAdd(itr, Prams(i), DateS)
    End If
  Next i
End Sub
```

Подпрограмма DateIntervals возвращает наибольший полный интервал указанного вами типа (год, месяц, день, час, минута, секунда) между двумя датами. Например, чтобы получить интервал времени в часах и минутах между 09:00 и 17:15, передайте в подпрограмму эти две даты, а также две переменные, задающие размерность интервала. Используйте запятые, чтобы пропустить более крупные ненужные интервалы:

```
Dim Hours As Variant, Minutes As Variant
Call DateIntervals(Now, "23.05.2000", , , , Hours, Minutes)
MsgBox "Часов = " & Hours & _
       "Минут = " & Minutes
```

Подпрограмма вернет "Часов = 8, Минут = 15".

Однако здесь следует обратить внимание на такой любопытный момент. Если вы выполните такое обращение к функции:

```
Call DateIntervals("28.02.2000", "01.03.2001", Years, , Days)
MsgBox Years & " " & Days
Call DateIntervals("29.02.2000", "01.03.2001", Years, , Days)
MsgBox Years & " " & Days
```

то получите для разных начальных дат один и тот же результат — 1 год и 1 день. Казалось бы, в подпрограмме есть ошибка, но это не так. Данный парадокс объясняется неопределенностью интервала в один год — он может быть 365 и 366 дней (так же, как и в один месяц). Соответственно в первом случае "год" является високосным (366 дней), а во втором — обычным (365 дней). Чтобы представить эту ситуацию, вообразите, что ваш знакомый говорит 31 января: "Позвони мне ровно через месяц" (или 29 февраля 2000 года — "ровно через год"). Когда же будет эта точная дата намеченного звонка?

Отметим также, что алгоритм расчета интервала с использованием привычных единиц можно выполнить и по-другому — сначала определить интервал в секундах, а потом

выделить из него минуты, часы и сутки (с месяцами и годами тут возникнут те же проблемы). Но он будет выглядеть не так изящно, как приведенная выше подпрограмма.

В процедуре DateIntervals хотелось бы обратить внимание еще на три используемые нами конструкции:

1. Для передачи возвращаемых параметров мы используем массив Param() с ключевым словом ParamArray. Такая конструкция применима только в конце списка аргументов подпрограммы и указывает, что данный аргумент является массивом типа Optional элементов типа Variant. С его помощью можно задать произвольное число аргументов. Кроме того, ParamArray нельзя использовать вместе с ключевыми словами ByVal, ByRef, или Optional.

   В принципе, можно было бы просто зарезервировать в вызывающей подпрограмме массив Param (0 To 5) и использовать непосредственно его. Но в данном случае подпрограмма выполнила бы расчет для всех элементов этого массива. Применение ParamArray позволяет нам пропускать "ненужные" параметры. Например, в нашем обращении мы получим результат в полных годах и весь остаток интервала — в секундах:

   ```
   Call DateIntervals("28.02.2000", "01.03.2001", Years, , , , ,Secs)
   ```

2. Для выбора нужного значения из двух вариантов мы используем функцию:

```
3.   MyVal = IIf(expr, truepart, falsepart)
```

   которая равнозначна такому варианту:

   ```
   If expr Then
     MyVal = truepart
   Else
     MyVal = falsepart
   End If
   ```

4. При коррекции временного интервала мы использовали такую конструкцию:

```
5.   If DateAdd(itr, Prams(i), DateS) <= DateF Then _
6.                   Prams(i) = Prams(i) - 1
```

   Любители хитроумных преобразований данных могли бы предложить более "изящный" вариант:

   ```
   Prams(i) = Prams(i) + (DateAdd(itr, Prams(i), DateS) > DateF)
   ```

   имея в виду, что арифметическое значение логического выражения будет равно -1 (True) или 0 (False). Мы, со своей стороны, настоятельно не рекомендуем пользоваться неявными преобразованиями типов данных.

Совет 289. Как правильно прочитать имя каталога

Проблема заключается в том, что в некоторых случаях имя каталога содержит символ обратной косой черты в конце, а иногда — нет. Пример такой путаницы — обращение к свойству Path объекта App. Если приложение находится в подкаталоге, то такой черты в конце не будет (например, C:\dir_x\dir_y\dir_z), но если оно располагается в корневом каталоге диска, то черта появится (например, C:\). Это нужно, в частности, учитывать при формировании полного имени файла, то есть вместо:

```
strFullFileName = App.Path & strFileName
```

нужно применять, например, такую конструкцию:

```
strFullFileName = App.Path & IIf(Right$( _
     App.Path, 1) = "\", "", '\') & strFileName
```

Если вам лень каждый раз писать этот код, можете создать функцию AppPath:

```
Public Function AppPath() As String
  ' Замените App.Path на
  ' AppPath во всем тексте программы
  AppPath = App.Path & IIf(Right$(App.Path, 1) = "\", "", "\")
End Function
```

Обратите внимание, что если вы автоматически добавляете обратную косую черту ко всем вызовам App.Path, а путь окажется корневым каталогом, то вы получите совершенно неинформативное сообщение об ошибке: Run-time error '5': Invalid procedure call.

Андрей Колесов, Ольга Павлова

© Андрей Колесов, Ольга Павлова, 2000
Авторский вариант. Статья была опубликована с незначительной литературной правкой в журнале "КомпьютерПресс" N 7/2000, компакт-диск.

Совет 290. Как заполнить поля электронного сообщения

ShellExecute является одной из наиболее гибких функций в Win32 API. С ее помощью можно передавать любое имя файла, а если расширение файла связано с какой-либо из программ, зарегистрированных на машине пользователя, то запускается соответствующее приложение, которое выводит или проигрывает указанный файл.

Здесь мы покажем, как использовать функцию ShellExecute для отправки электронных сообщений. Вы сможете задавать не только адрес получателя, но и списки получателей (CC и BCC), тему и текст сообщения, а также вставлять файл или его часть в отправляемое сообщение. Для этого необходимо создать строковую переменную, добавить список основных адресов (отделенных друг от друга точкой с запятой) и знак вопроса:

```
для копий CC (Копия): &CC= (список получателей)
для невидимых (слепых) копий: &BCC= (список получателей)
для темы сообщения: &Subject= (тема сообщения)
для текста сообщения: &Body= (текст сообщения)
для присоединения файла: &Attach= (путь к файлу, заключенный в кавычки)
```

Продемонстрируем это на примере. Создайте новый VB-проект, добавьте форму и разместите на ней шесть текстовых полей и одну командную кнопку cmdSendIt (см. рис. 290).

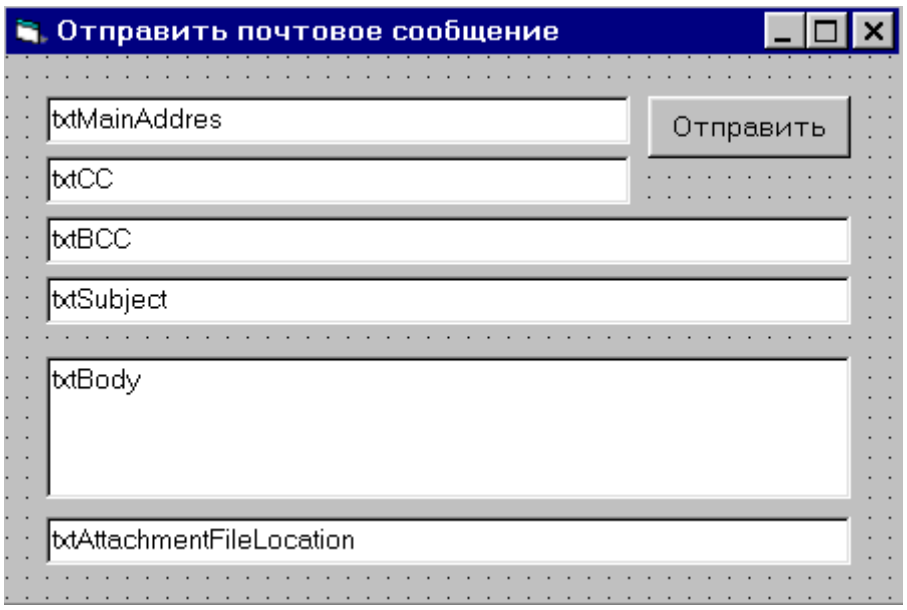


Рис. 290

Напишите такой код в разделе Declarations:

```
Private Declare Function ShellExecute Lib _
  "shell32.dll" Alias "ShellExecute" _
     (ByVal hWnd As Long, ByVal lpOperation As _
     String, ByVal lpFile As String, ByVal _
     lpParameters As String, ByVal lpDirectory _
     As String, ByVal nShowCmd As Long) As Long
Private Const SW_SHOWNORMAL = 1
```

Затем введите следующий код для события Click командной кнопки:

```
Private Sub cmdSendIt_Click()
  Dim sText As String
  Dim sAddedText As String
  If Len(txtMainAddresses) Then
       sText = txtMainAddresses
  End If
  If Len(txtCC) Then sAddedText = sAddedText & "&CC=" & txtCC
  If Len(txtBCC) Then sAddedText = sAddedText & "&BCC=" & txtBCC
  If Len(txtSubject) Then _
       sAddedText = sAddedText & "&Subject=" & txtSubject
  If Len(txtBody) Then _
       sAddedText = sAddedText & "&Body=" & txtBody
  If Len(txtAttachementFileLocation) Then _
       sAddedText = sAddedText & "&Attach=" & Chr(34) & _
            txtAttachementFileLocation & Chr(34)
  sText = "mailto:" & sText
  If Len(sAddedText) <> 0 Then Mid$(sAddedText, 1, 1) = "?"
  sText = sText & sAddedText
  If Len(sText) Then _
    Call ShellExecute(Me.hWnd, "open", sText, _
     vbNullString, vbNullString, SW_SHOWNORMAL)
End Sub
```

Здесь следует обратить внимание на два момента:

1. Между знаками "амперсанд" (&) и "тэг" или знаками "тэг" и "равно" не нужно ставить пробелы.
2. Из-за отсутствия возможностей форматирования текст сообщения будет состоять из одного параграфа.

Тем не менее, используя предложенный здесь способ, вы сможете создавать работающие почтовые аплеты всего за несколько секунд.

Примечание. *Полная функциональность полей электронного сообщения может быть достигнута только в почтовых клиентах, совместимых с Microsoft Exchange. С другими почтовыми клиентами некоторые или даже все эти поля могут не работать.*

Совет 291. Как открыть VB-файл в Notepad или в любом другом текстовом редакторе

Предположим, вы хотите сделать так, чтобы с помощью щелчка правой кнопки мыши можно было открыть VB-файл в редакторе Notepad и скопировать фрагмент кода для другого приложения, например для того, над которым вы сейчас работаете. Попробуйте сделать следующее.

Создайте текстовый файл с именем FormEdit.reg, введите в него такой код, сохраните, а затем закройте его:

```
REGEDIT4
[HKEY_CLASSES_ROOT\VisualBasic.Form\shell\edit]
@="&Edit"
[HKEY_CLASSES_ROOT\VisualBasic.Form\shell\edit\command]
@="C:\Windows\Notepad.exe %1"
```

Дважды щелкните мышью на этом файле, и его содержимое автоматически загрузимся в Системный Реестр. Теперь щелкните правой кнопкой мыши на любом FRM-файле, и в появившемся "быстром" меню вы увидите команду Edit. Выполните ту же самую операцию для других текстовых VB-файлов — VisualBasic.ClassModule, VisualBasic.Module и VisualBasic.Project, а затем с помощью программы Regedit.exe в Windows проверьте полученные результаты.

Если вместо Notepad вы хотите использовать Word или любой другой текстовый редактор, напишите примерно такую командную строку (для Word):

```
@="c:\\Program Files\\Microsoft Office\\Office\\Winword.exe %1"
```

Здесь следует быть особенно внимательными — используйте две обратные косые черты и, кроме того, не ошибитесь, указывая полный путь к файлу Winword.exe.

Хотя VB4, VB5 и VB6 имеют различные точки входа в Реестре, подобную методику можно применять для любого текстового VB-файла, включая BAS, CLS, FRM и VBP. Она не подходит для FRX- или других двоичных файлов, но с ее помощью можно реализовать просмотр HTML-файлов. Указав путь к используемому по умолчанию браузеру, можно, щелкнув правой кнопкой мыши на любом HTML-файле, редактировать его как простой текст. Кроме того, можно настроить браузер на просмотр GIF-файлов, чтобы увидеть, что они собой представляют. Но самое главное — всякий раз, работая с Реестром, будьте крайне осторожны.

Совет 292. Как ускорить операцию деления чисел с плавающей запятой

Если вам приходится выполнять много арифметических операций деления чисел с плавающей запятой в VB, можете попробовать оптимизировать эти действия, осуществляя умножение на обратную величину. Например, вместо операции:

```
X / Y
```

Выполните:

```
X * (1 / Y)
```

Чтобы увидеть, как это работает на практике, создайте новый проект в VB и введите следующий код в событие Form_Click:

```
Private Declare Function GetTickCount _
     Lib "kernel32" () As Long
Const NORMAL As Double = 1453
Const RECIPROCAL As Double = 1 / NORMAL
Const TOTAL_COUNT As Long = 10000000
Private Sub Form_Click()
  Dim dblRes As Double
  Dim lngC As Long
  Dim lngStart As Long
  '
  On Error GoTo Error_Normal
  '
  lngStart = GetTickCount
  For lngC = 1 To TOTAL_COUNT
    dblRes = Rnd / NORMAL
  Next lngC
  MsgBox "Обычное время: " & GetTickCount - lngStart

  lngStart = GetTickCount
  For lngC = 1 To TOTAL_COUNT
    dblRes = Rnd * RECIPROCAL
  Next lngC
  MsgBox "Время для обратной величины: " _
     & GetTickCount - lngStart
Exit Sub
Error_Normal:
  MsgBox Err.Number & " - " & Err.Description
End Sub
```

Вы обнаружите, что скорость выполнения вычислений увеличится примерно на 15% при использовании метода умножения на обратную величину. Тем не менее будьте осторожны при округлении чисел — так, 3 / 3 = 1, но 3 * (0,333333...) = 0,999999....

Совет 293. Как ускорить выполнение операций ввода-вывода для файлов

Довольно часто в программах встречаются ситуации, когда нужно просто целиком скопировать содержимое файла в оперативную память. Например, это необходимо для выполнения копирования файлов или операций быстрого поиска контекста.

Для текстовых файлов можно предложить такие две процедуры чтения и записи файлов:

```
Public Function ReadFile(FileName _
```

```
       As String) As String
  Dim FileNumber As Integer
  FileNumber = FreeFile
  Open FileName For Input As #FileNumber
  ReadFile = Input(LOF(FileNumber), FileNumber)
  Close #FileNumber
End Function
Public Sub WriteFile(FileName As _
       String, Contents As String)
  Dim FileNumber As Integer
  FileNumber = FreeFile
  Open FileName For Output As #FileNumber
  Print #FileNumber, Contents;
  Close #FileNumber
End Sub
```

Обратите внимание, что перед открытием файлов мы получаем свободный номер файла с помощью функции FreeFile. С применением таких функций операция копирования двух файлов принимает такой вид:

```
Call WriteFile("c:\b.txt", ReadFile("c:\a.txt"))
```

Для копирования файлов произвольного формата следует использовать тип файлов Binary. Тогда операции чтения-записи будут выглядеть так:

```
Open FileInput$ For Binary As #FileInp
FileString$ = Space$(LOF(FileInp))
Get #FileInp,, FileString$
ReadFile = FileString$
...
Open FileOutput$ For Bi As #FileOut
Put #FileOut,, FileString$
```

Но в этом случае нужно сначала сделать проверку — может быть, выходной файл уже существует (если это так, то в него просто перепишутся первые Len(FileString$) символов).

Однако следует иметь в виду, что для точного чтения содержимого произвольных файлов использование строковой переменной не очень хорошо подходит. Дело в том, что в этом случае при чтении производится преобразование байтов из однобайтовой ANSII-кодировки в двухбайтовую Unicode (подробнее об этом см. КомпьютерПресс N 10'99, CD-ROM). При записи производится обратное преобразование.

Поэтому для создания точной копии содержимого файлов следует использовать байтовый динамический массив:

```
ReDim ArrByte (1 To LOF(FileNumber))
Get #FileInp,, ArrByte
...
Get #FileOut,, ArrByte
```

Совет 294. Как создать дополнение, закрывающее все окна проекта

При создании VB-проекта вы наверняка открываете множество различных окон, особенно выполняя операции поиска/замены внутри программного кода. В больших проектах количество открытых окон вырастает настолько, что становится обременительным

закрывать их одно за другим. Поэтому мы предлагаем создать простое дополнение, которое бы выполняло эту работу за вас.

Создайте новый проект типа AddIn (значок Addin в окне New Project). Вы получите шаблон MyAddIn, в котором содержится некоторый программный код для построения дополнения. В код формы frmAddIn введите следующее:

```
Private Sub Form_Load()
  Dim w As Window
  For Each w In VBInstance.Windows
    ' закрывает все видимые
    ' окна кода и форм
    If (w.Type = vbext_wt_CodeWindow _
      Or w.Type = vbext_wt_Designer) _
       And w.Visible Then w.Close
    End If
  Next
  Unload Me
End Sub
```

В окне Object Browser щелкните правой кнопкой мыши на проекте MyAddIn. В появившемся "быстром меню" выберите команду Properties и измените имя и описание дополнения. Затем в коде шаблона замените My Add-In на то имя, которое бы вы хотели видеть в меню Add-Ins в среде разработки VBE. Если вы работаете в VB6, то помимо этого необходимо еще поменять имя дополнения в конструкторе AddInDesigner. Создайте DLL-библиотеку (команда File|Make MyAddIn.dll), и ваше дополнение должно появиться в списке в Add-In Manager. Добавьте его к командам меню Add-Ins, а затем запустите — все открытые окна проекта закроются в одно мгновение!

Совет 295. Используйте ключевое слово WithEvents для связи форм MDI и MDIChild

Здесь приводится изящный способ передачи событий, таких как щелчки на панели инструментов или выделение команд меню, из родительской MDI-формы в активную дочернюю MDIChild-форму в многодокументном приложении. Предположим, что MDI-форма содержит элемент управления Toolbar с именем tbrMain, для которого введите следующий код:

```
Event ButtonClick(strKey As String)
Private Sub tbrMain_ButtonClick(ByVal _
     Button As MSComctlLib.Button)
     RaiseEvent ButtonClick(Button.Key)
End Sub
```

Затем напишите такой код для каждой MDIChild-формы, которая должна получить событие ButtonClick:

```
Private WithEvents m_mdiParent As mdiParent
Private Sub tbrMain_ButtonClick(ByVal _
  Button As MSComctlLib.Button)
  RaiseEvent ButtonClick(Button.Key)
End Sub

Private Sub Form_Acitivate()
  Set m_mdiParent = mdiParent
End Sub
```

```
Private Sub Form_Deactivate()
  Set m_mdiParent = Nothing
End Sub

Private Sub m_mdiParent_ButtonClick (strKey As String)
  ' Пример кода, в котором значения
  ' Button.Key соответствуют кнопкам
  ' New, Change, Delete и Save
  Select Case strKey
    Case "New"
      PerformNewAction
    Case "Change"
      PerformChangeAction
    Case "Delete"
      PerformDeleteAction
    Case "Save"
      PerformSaveAction
  End Select
End Sub
```

Использование этой подпрограммы аналогично объявлению элемента управления с именем m_mdiParent, у которого есть событие ButtonClick. Используйте события Activate и Deactivate, чтобы форма MDIChild являлась единственной, которая бы получала событие ButtonClick.

Совет 296. Помните об операторе Option Base

В общем случае при резервировании массива нужно указывать в явном виде его нижнюю и верхнюю границу индекса:

```
ReDim MyArray (LowBoundary To UpBoundary)
```

Но довольно часто мы используем более простую конструкцию, когда значение нижней границы задается по умолчанию:

```
ReDim MyArray (UpBoundary)
Dim MyArrayStat(100)
```

Но чему же равна нижняя граница и сколько элементов на самом деле содержится в массиве?

В языках программирования используется два варианта соглашений: например, в Фортране — нумерация традиционно начинается с единицы, а в Бейсике — с нуля. Во всех версиях Microsoft Basic (Quick for DOS, Visual for Windows) программист может управлять установкой значения нижней границы по умолчанию с помощью оператора Option Base. Он помещается в секции Declarations в начале программного модуля (форма, модуль кода, модуль класса и пр.) и его действие распространяется на все объявления массивов внутри модуля.

Таким образом, значение нижней границы для приведенного выше примера Dim MyArrayStat(100) будет следующим:

- = 0 — если оператор Option Base отсутствует;
- = 0 — задан Option Base 0;
- = 1 — задан Option Base 1.

Все это нужно иметь в виду, если вы вставляете фрагмент программного кода в какие-либо модули (например, из своей библиотеки повторно используемых процедур), — значение нижней границы будет зависеть от наличия в данном модуле оператора Option Base.

Совет: *Если для программы принципиально важно наличие или отсутствие нулевого индекса, то указывайте обе границы индекса в явном виде.*

Еще одно замечание. В VB 5/6 имеется возможность автоматического создания и заполнения байтового массива при перезаписи в него строковой переменной:

```
Dim arrByte () As Byte
Source$ = "Строковая переменная"
arrByte = Source$
```

В этом случае всегда создается массив размерностью от 0 до (LenB(Source$)-1) независимо от наличия/отсутствия оператора Option Base.

Совет 297. Экспорт таблиц в виде текстовых файлов

Здесь приведена простая процедура, которая выводит в виде текстового файла записи из таблицы базы данных или из таблицы, сформированной в результате SQL-запроса. Эта информация потом может быть считана любым текстовым редактором или программой электронных таблиц. В этом примере Db — это глобальная переменная объекта, которая должна быть до обращения к процедуре определена как база данных. sSource — имя таблицы или SQL-запрос. Кроме имени выходного файла пользователь должен также задать код разделителя полей записи в текстовом файле:

```
Public Function TableToSpreadsheetMy(sSource As String, _
  sFile As String, sSeparator As String) As Boolean
  ' Включение обработки ошибок
  On Error GoTo TableToSpreadsheet_Err
  '
  ' Синтаксис обращения:
  ' If TableToSpreadsheet("SELECT * FROM _
  ' Authors", "C:\Temp\Authors.csv", Chr$(9)) = True _
  ' Then....
  '
  Dim rsTemp As Recordset
  Dim sHeader As String
  Dim sRow As String
  Dim i As Integer, nFile As Integer
  ' Формирование набора данных
  ' Глобальная объект-переменная Db должна
  ' быть открыта ранее как база данных
  Set rsTemp = Db.OpenRecordset(sSource)
  With rsTemp
    TableToSpreadsheet = False
    ' Есть ли записи в наборе?
    If .RecordCount > 0 Then
      ' Создание выходного файла
      nFile = FreeFile
      Open sFile For Output As #nFile
      ' Вывод имен полей таблицы
      sHeader = .Fields(0).Name
      If .Field.Count > 0 Then
        For i = 1 To .Fields.Count - 1
          sHeader = sHeader & sSeparator & .Fields(i).Name
```

```
        Next i
      End If
      Print #nFile, sHeader
      ' Вывод содержимого полей таблицы
      .MoveFirst
      Do Until .EOF
        sRow = .Fields(0).Value
        If .Field.Count > 0 Then
          For i = 1 To .Fields.Count - 1
            sRow = sRow & sSeparator & .Fields(i).Value
          Next i
        End If
        Print #nFile, sRow
        .MoveNext
      Loop
      Close #nFile ' Target file is complete
      TableToSpreadsheet = True
    End If
    .Close
  End With
  Set rsTemp = Nothing ' закрываем временный объект
  Exit Function
TableToSpreadsheet_Err:
  LogIt "TableToSpreadsheet : " & Err.Description
  ' Запись информации об ошибке
  Resume Next
End Function
```

Андрей Колесов, Ольга Павлова

© 2000, Андрей Колесов, Ольга Павлова
Авторский вариант. Статья была опубликована с незначительной литературной правкой в журнале "КомпьютерПресс" N 8/2000, CD-ROM.

Совет 298. Быстрый своппинг строковых переменных

Для обмена данными между двумя строковыми переменными можно использовать такую простую процедуру:

```
Sub SwapString (String1$, String2$)
  Dim Save$
  Save = String1$ ' запись в промежуточную переменную
  String1$ = String2$
  String2$ = Save$
End If
```

Однако мы уже неоднократно подчеркивали, что операции присвоения строковых переменных требуют довольно много времени, так как это связано с динамическим перерезервированием оперативной памяти. Кроме того, время их выполнения напрямую зависит от длины строк.

Мы предлагаем "хитрый" вариант взаимного обмена содержимого двух строк, который выполняется очень быстро и не зависит от числа байтов в строке. Он основан на использовании API-функции для копирования областей памяти и применении реально существующих, но не описанных в документации функций VB (так называемые недокументированные функции):

```
Declare Sub CopyMemory Lib "kernel32" Alias _
  "RtlMoveMemory" (Destination As Any, _
  Source As Any, ByVal Length As Long)

Public Sub SwapSrting(String1$, String2$)
  Dim Save As Long
  'Своппинг двух строковых переменных
  Save = StrPtr(String1)
  Call CopyMemory(ByVal VarPtr(String1), ByVal VarPtr(String2), 4)
  Call CopyMemory(ByVal VarPtr(String2), Save, 4)
End Sub
```

Как известно, в VB строковые переменные имеют описатель (дескриптор) и собственно содержимое строки. В данном случае мы выполняем своппирование не содержимого строк, а содержимого описателей. Для этого используются две недокументированные VB-функции:

- StrPrt — возвращает четырехбайтовый адрес содержимого строки;
- VarPrt — возвращает четырехбайтовый адрес дескриптора.

Здесь нужно отметить два момента:

1. Такой прием можно использовать только для своппинга — операцию присвоения строковой переменной таким образом делать нельзя (недопустимо, чтобы одна и та же строка данных использовалась в двух разных дескрипторах).
2. В VB описатель строки содержит только адрес ее содержимого. Число байтов строки находится в четырех байтах, которые хранятся непосредственно перед этим содержимым, то есть функция LenB(MyString$) может быть реализована в виде такой конструкции:

```
3.   Dim LenBmy&
4.   Call CopyMemory(LenBmy$, ByVal StrPtr(MyString$) - 4, 4)
```

Тем, кто еще пользуется QuickBasic для DOS, напомним, что здесь длина строки хранится в ее описателе вместе с адресом. Но нужно иметь в виду, что адресация строк в QB 4.x и PDS 7.x различается, и соответственно различаются описатели строк. Адрес дескриптора можно получить командами VARSEG (адрес сегмента) и VARPRT (смещение внутри сегмента); копирование областей памяти производится командами PEEK (чтение байта) и POKE (запись байта).

Совет 299. Динамическое управление меню

Многие программисты порой забывают о том, что меню, создаваемое с помощью Menu Editor, состоит из компонентов, которые работают так же, как и все остальные элементы управления. Это, в частности, позволяет непосредственно в процессе работы приложения управлять пользовательским интерфейсом. Так, можно изменять название команды меню (свойство Caption), делать ее недоступной для выполнения (Enable) или невидимой (Visible). Если же вы создали массив меню, назначив данному компоненту индекс (свойство Index), динамически создавать или удалять элементы такого массива, можно, например, следующим образом:

```
i% = mnuFileArray.Ubound + 1  ' следующий номер индекса после самого
                              ' большого уже существующего
Load mnuFileArray(i%)  ' создание нового элемента массива меню
mnuFileArray(i%).Caption = "NewArray" & i%  '  название команды
...
Unload mnuFileArray(1)        ' удалить элемент с индексом 1
```

Совет 300. Используйте каталог Template

Если у вас есть VB-проект, который вы хотите хотя бы иногда использовать в качестве шаблона при создании новых проектов, поместите его в каталог ...\VisualStudio\VB98\Template\Projects\ или другой каталог, который вы указали при установке VB 6.0. (Для VB 5.0 этот каталог называется VB5\Template\Projects\.) Тогда ваш шаблон будет автоматически появляться в окне New Project при создании нового проекта.

Такой вариант создания нового проекта на основе шаблона лучше, чем простая загрузка существующего проекта командой Open, ибо в последнем случае есть опасность забыть, что необходимо обязательно сразу же сохранить проект и все его компоненты командой Save As, иначе все изменения проекта будут записываться в исходный шаблон.

Точно так же вы можете записать шаблоны для отдельных компонентов проекта, которые хранятся в подкаталоге Template:

```
Template/Подкаталог   При выполнении команды Project/Add
                      компонент
-----------------------------------
FORMS                 Form
MDIForms              MDI Form
MODULES               Modules
CLASSES               Class Modules
USERCTLS              User Control
PROPPAGE              Property Page
USERDOCS              User Document
-----------------------------------
```

Совет 301. Используйте Microsoft Forms 2.0 Form (когда это нужно)

Если вы хотите создавать программные компоненты формы (модули формы), которые можно использовать как в VB, так и в Office/VBA, то вам нужно пользоваться конструктором MS Forms 2.0. В среде VBA он вызывается командой Insert|UserForm, в среде VB — Project|Add Microsoft Forms 2.0 Form (если такая команда отсутствует в среде VB, то конструктор нужно подключить с помощью команды Project|Components|Designers).

Напомним, что формы UserForms в VBA — это не то же самое, что VB-формы (Ruby Forms). VBA UserForms являются экземплярами ActiveX-конструктора Microsoft Forms 2.0 (FM20.dll), который входит в состав как VB, так и VBA. Более того, все элементы управления, представленные в нем по умолчанию, являются не встроенными, а ActiveX-компонентами (то есть при желании их можно подключить и к VB-форме). Чтобы убедиться в этом, откройте окно Tools|Additional Controls.

К сожалению, эти два вида форм различаются не только форматом их модулей. Каждая из них использует свой собственный набор встроенных элементов управления. Проблема же заключается в том, что эти наборы не только не совпадают функционально, но даже для одинаковых по значению элементов управления используются разные наименования свойств, событий и методов. MS Forms 2.0 не поддерживают ряд очень полезных встроенных элементов управления VB (Timer, FileListBox, PictureBox, DriveListBox, DirListBox, Menu, Shape и Line), но при этом включают другие полезные компоненты (TabStrip, MultiPage) и команды проектирования формы (TabOrder).

Совет 302. Используйте элемент управления IE Timer

Отсутствие встроенных элементов управления (на что мы сетовали в предыдущем совете) можно компенсировать возможностью использования дополнительных элементов управления ActiveX. Если у вас имеется VB 5.0 или 6.0, то вы можете самостоятельно сделать, например, на основе встроенного элемента Timer собственный ActiveX-компонент для реализации функции таймера. Такой OCX можно было бы использовать в конструкторе MS Forms 2.0 (VBA-формы). Но прежде чем делать свои компоненты, стоит посмотреть, нет ли уже готового подходящего элемента на Web-сайтах с VB-ресурсами и разными Freeware & Shareware.

Так, очень нужный порой компонент Timer в виде файла IETimer.OCX можно найти по адресу www.visual.2000.ru/develop/vb/source/[49].

Но здесь мы хотели бы отметить некоторые различия в процедурах подключения OCX к среде VB и VBA.

1. Обычно OCX-компоненты помещаются в системный каталог Windows/System/. Однако мы рекомендуем размещать такие дополнительные файлы в каком-нибудь специально созданном для этого пользовательском подкаталоге. Это упростит процедуры "чистки мусора" и обновления версий компонентов, так как можно будет четко следить за тем, что вы записали сами, а что автоматически прописали в System операционная система и программы установок разных приложений.
2. Далее необходимо зарегистрировать OCX-файл. Лучше всего это сделать с помощью утилиты RegSvr32.exe, которая входит в состав VB и Office (проще всего скопировать ее в подкаталог с вашими OCX):
3. `RegSvr32.exe Your.OCX`
4. Подключение к панели Toolbox для VB- и VBA-форм несколько различается.
    - Для VB-форм (в среде VB) выберите команду Project|Components, а потом во вкладке Controls установите флажок IE Timer (в данном случае зарегистрировать OCX можно, подключив нужный файл с помощью кнопки Browse). В среде Office/VBA также имеется команда (и диалоговое окно) Insert|Components|Controls, но зачем она здесь присутствует, куда подключаются выделенные элементы управления, — этого мы не поняли.
    - Для подключения дополнительного элемента управления в среде VBA выберите команду Tools|Additional Controls. К ней можно также обратиться с помощью контекстного меню, которое появляется после щелчка правой кнопкой мыши панели Toolbox (только так и можно сделать, работая с MS Forms 2.0 в среде VB). Но проблема в данном случае заключается в том, что имя искомого компонента будет другим — Timer Object.
5. Не менее интересна процедура отключения дополнительного компонента. Для VB-форм нужно войти в окно Components|Controls и убрать в нем флажок. Для VBA-форм эту же операцию можно сделать в окне Additional Controls. Только проще — щелкнуть правой кнопкой мыши на изображении компонента на панели Toolbox и выбрать команду Delete. Кстати, с помощью этого же меню в данном случае можно изменить название и изображение пиктограммы элемента управления (рис. 302):

[49] http://www.visual.2000.ru/develop/vb/source/index.htm

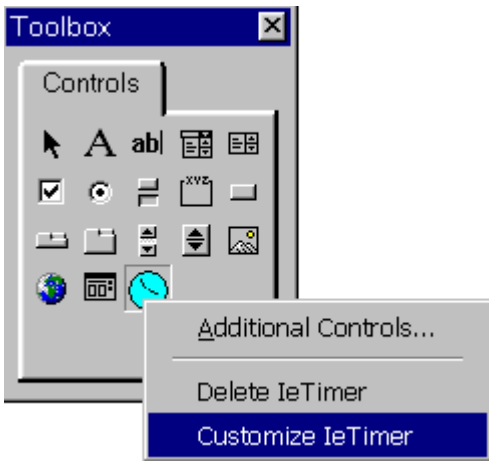


Рис. 302

6. Еще один вопрос — как убрать имя ненужного компонента из списков в окнах Components|Controls и Additional Controls. Отменить регистрацию OCX (вернее, любого ActiveX-компонента) можно с помощью той же утилиты RegSvr32:

```
7.  RegSvr32.exe /u Your.OCX
```

Но иногда оказывается, что этого недостаточно, и приходится вручную просматривать (искать заданный файл) и чистить Реестр.

Совет 303. Как погрузить приложение в "глубокий сон"

Мы уже приводили несколько советов, как можно приостановить выполнение приложения на некоторое время. В частности, это можно сделать с помощью элемента управления Timer или путем опроса текущего времени (обращения к функции Timer) в цикле программы. Однако оба варианта имеют некоторые ограничения. Например, элемент управления Timer позволяет делать задержку в диапазоне от 1 секунды до 1 минуты, а функция Timer имеет минимальную дискретность в 1 секунду. При использовании соответствующей API-функции этих ограничений нет:

```
' описание функции
Declare Sub Sleep Lib "kernel32" (ByVal dwMilliseconds As Long)
' обращение к ней (аргумент задается в миллисекундах)
Sleep 150010 ' ожидание 150,01 секунды
```

Но при использовании этой конструкции есть другая проблема — "глубокий сон" приложения не удастся прервать до истечения заданного интервала времени.

Совет 304. Управление расстоянием между столбцами в ListBox

С помощью API-функции SendMessage вы можете управлять позициями табулятора в элементе управления ListBox. Это может быть полезно, если каждая строка списка состоит из полей, разделенных символом Tab (код ASCII = 9), то есть если список выводится в виде таблицы, содержащей несколько колонок. Для этого в программный модуль запишите такое объявление и подпрограмму:

```
Private Declare Function SendMessage Lib _
    "user32" Alias "SendMessageA" _
    (ByVal hWnd As Long, ByVal wMsg As Long, _
    ByVal wParam As Long, lParam As Any) As Long
```

```
Private Const LB_SETTABSTOPS = &H192
Public Sub SetListTabStops(MyListBox As ListBox, _
              ParamArray ParmList() As Variant)
  'Коррекция позиции Tabs в списке ListBox
  '
  ' Для передачи переменного числа однородных параметров
  ' используется конструкция ParamArray
  ' ВНИМАНИЕ! Нижний индекс ParmList равен 0
  ' даже при Option Base 1!!!
  ' Позиция табулятора определяется в специальных величинах
  ' окна, которая в среднем равна 1/4 символа

  Dim i As Long
  Dim NumColumns As Long
  '
  ' формирование массива для установки табуляторов
  ReDim ListTabs(0 To UBound(ParmList)) As Long
  For i = 0 To UBound(ParmList)

          ListTabs(i) = ParmList(i)
  Next i
  NumColumns = UBound(ParmList) + 1
  '
  ' установка новых значений позиций табулятора
  Call SendMessage(MyListBox.hWnd, LB_SETTABSTOPS, _
   NumColumns, ListTabs(0))
  ' вывести новое изображение списка
  lstMyListBox.Refresh
End Sub
```

Чтобы протестировать эту конструкцию, создайте форму, на которой разместите элемент списка lstMyListBox и две командные кнопки Command1 и Command2. Далее запишите такой программный код для этих компонентов:

```
Private Sub Form_Load()
  ' Начальное формирование списка
  ' с тремя колонками, разделенными символом vbTab = Chr$(9)
  lstMyListBox.AddItem "Колонка11" & vbTab & "Колонка12" _
         & vbTab & "Колонка13"
  lstMyListBox.AddItem "C21" & vbTab & "C22" _
         &  vbTab & "C23"
End Sub

Private Sub Command1_Click()
  ' одинаковое расстояние в 52 позиции (13 символов)
  Call SetListTabStops(lstMyListBox, 52)
End Sub

Private Sub Command2_Click()
  ' переменное расстояние (10 и 30 символов)
  Call SetListTabStops(lstMyListBox, 40, 120)
End Sub
```

Запустите на выполнение созданный проект, и вы увидите такое изображение списка:

```
Колонка11       Колонка12       Колонка13
C21     C22     C23
```

Щелкните кнопку Command1 — список примет следующий вид:

```
Колонка11     Колонка12     Колонка13
C21           C22           C23
```

Щелкните Command2, и получится другое изображение:

```
Колонка11 Колонка12           Колонка13
C21       C22                 C23
```

Андрей Колесов, Ольга Павлова

© 2000, Андрей Колесов, Ольга Павлова
Авторский вариант. Статья была опубликована с незначительной литературной правкой в журнале "КомпьютерПресс" N 8/2000, CD-ROM.

Совет 305. Где хранятся настройки панелей инструментов MS Office

Как известно, пользователь MS Office 97/2000 может выполнять различные операции по настройке меню и панелей инструментов среды — модифицировать, удалять и добавлять команды и панели. Это можно делать как с помощью команды Настройка (Customize), так и программным образом (VBA). Но где хранятся эти настройки, как они связаны с обрабатываемыми документами? Это полезно знать, особенно если вы хотите перенести эти настройки на другой компьютер. Тем более что реализация механизма сохранения настроек для всех офисных приложений различная (это к вопросу о том, что MS Office представляет собой как бы единое семейство продуктов).

Word

Это приложение обладает наиболее удобной и гибкой системой настроек. Как известно, VBA-проекты хранятся в документах или шаблонах Word. В них же хранятся и настройки панелей инструментов, и пользователь может сам выбирать, в каком именно файле они будут записаны.

Таким образом можно реализовать самые разнообразные варианты подключения расширений пользовательской среды. Если записать эти настройки в глобальный шаблон (например, в Normal.dot), то данные команды будут доступны при работе со всеми документами в данной сессии. При сохранении установок в конкретный документ они будут применяться только в нем. Если же настройки включены в присоединенный шаблон, они будут появляться во всех документах, которые его используют.

Excel и PowerPoint

В этих двух приложениях макрокод хранится в документах (рабочих книгах и презентациях), а сами настройки панелей инструментов — в фиксированных файлах. Для версии 97 это соответственно файлы <username>.xlb и <username>.pcb в каталоге Windows, для версии 2000 — файлы Excel\Excel.xlb и PowerPoint\ppt.pcb, которые находятся в каталоге Windows\Application Data\Microsoft\. Таким образом, получается, что пользователь может работать только с одним фиксированным вариантом настроек панелей инструментов.

Но в этих приложения механизм работы команд несколько различается. Для примера напишите такую макрокоманду:

```
Sub Hello
    MsgBox "Hello!"
End Sub
```

которую запишите в книгу Hello.xls и презентацию Hello.ppt.

Соответственно в Excel и PowerPoint сделайте кнопку на панели инструментов для обращения к этим макрокомандам. Затем перезагрузите Excel: на панели инструментов будет видна ваша кнопка, хотя программный код для нее не загружен. Нажмите кнопку — произойдет загрузка Hello.xls и выполнится макрокоманда. Теперь повторите те же действия для PowerPoint — макрокоманда выполнится, но презентация не загрузится в среду приложения.

В Excel можно скопировать созданные пользователем панели инструментов в рабочую книгу (со встроенными это проделать нельзя!). Это делается с помощью диалогового окна Attach Toolbars (вызывается командой Attach на вкладке ToolBars окна Customize) — программным образом с применением VBA это сделать нельзя.

После копирования можно удалить пользовательские панели из среды самого Excel с помощью команды Delete окна Customize или метода Delete коллекции CommandBars. Удалить панель из рабочей книги можно только с помощью диалогового окна.

Важное замечание. *Когда вы открываете рабочую книгу, производится автоматическое копирование пользовательских панелей из книги в среду Excel (но только в том случае, если панелей с такими именами не было в самом приложении!). При этом следует иметь в виду, что при закрытии рабочей книги эти панели остаются в Excel. Поэтому, если вы хотите, чтобы подобные компоненты появлялись только в сеансе работы с данной рабочей книгой, вы можете программным образом сделать удаление панели при закрытии книги. Если же вы не удалите пользовательскую панель из Excel и будете корректировать ее, то в последующих сеансах вы будете работать именно с ней (копирование из рабочей книги не будет выполняться).*

Outlook и FrontPage

В этих двух приложениях все сделано еще проще: программный код хранится в строго фиксированных проектах — Outlook\VbaProject.OTM и FrontPage\Macros\Microsoft FrontPage.fpm, а настройки среды — в столь же фиксированных Outlook\outcmd.dat и FrontPage\State\cmpUI.prt. Все это, в свою очередь, находится в подкаталоге Windows\Application Data\Microsoft\.

Примечание. *Для приложений Excel, PowerPoint, Outlook и FrontPage каталог Windows\Application Data\Microsoft\ используется только в случае однопользовательского режима. Если же в операционной системе установлен многопользовательский режим с использованием различных профилей, то в этом случае используется каталог Windows\Profiles\Username\Application Data\Microsoft\.*

Access

В документах Microsoft пользовательские настройки встроенных панелей инструментов среды Access хранятся в "специальной базе данных". Наши исследования показали, что эта база — Реестр Windows, хранящийся по адресу: HKCU\Software\Microsoft\Office\9.0\Access\Settings\Commandbars\. Но что и как туда пишется — нам разобраться не удалось. Панели же инструментов, создаваемые с помощью Access, хранятся в файлах этой базы данных.

Совет 306. Управление порядком перемещения по элементам управления

Как известно, перемещение фокуса от одного элемента управления к другому можно выполнять с помощью клавиатуры: Tab (вперед) или Shift+Tab (назад). Порядок передвижения определяется значением свойства TabIndex элементов управления, который начинается с нуля и для каждого размещаемого на форме элемента управления увеличивается на единицу. Свойство TabIndex, конечно же, имеется только у видимых компонентов (у Timer его нет). Пассивные визуальные компоненты типа Label имеют свойство TabIndex, но фактически не используют его — при перемещении по форме такие элементы автоматически пропускаются.

Для изменения последовательности обхода визуальных компонентов (она редко совпадает с тем порядком, в котором мы создавали диалоговый интерфейс) нужно менять свойства TabIndex. Это можно делать вручную, напрямую корректируя параметры в окне Properties. Обратите внимание, что ошибка использования одного и того же значения TabOrder для разных компонентов здесь исключена за счет автоматической перенумерации.

Пусть, например, у вас имеются четыре компонента с номерами 0, 1, 2, 3. Для последнего вы решили установить TabOrder=1. В этом случае автоматически изменятся значения свойств TabOrder для двух элементов управления: 1->2, 2->3.

Однако гораздо удобнее использовать специальные визуальные средства управления перемещения по элементам формы. Как мы отмечали в совете 301[50], конструктор MS Forms 2.0 (в среде VB и VBA) для этого использует специальную встроенную команду TabOrder. Чтобы воспользоваться ею, щелкните правой кнопкой мыши форму и в появившемся контекстом меню выберите команду Tab Order. В появившемся одноименном диалоговом окне выделите нужные компоненты и с помощью кнопок Move Up и Move Down передвиньте их в нужную позицию списка, который определяет порядок перемещения (рис. 306-1).

[50] http://www.visual.2000.ru/develop/ms-vb/tips/0008-1.htm

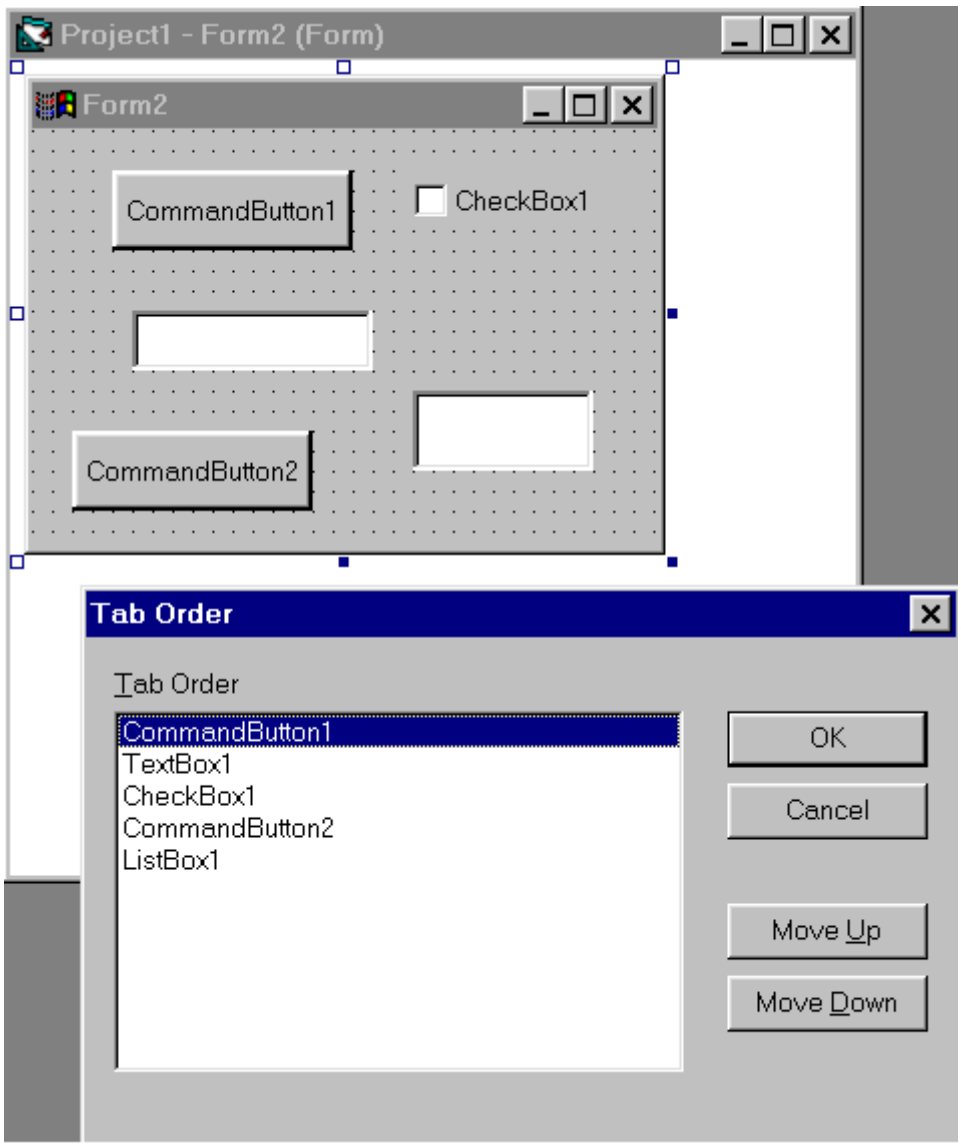


Рис. 306-1

Для обычных VB-форм такой встроенной команды нет, и по этому поводу мы уже выказывали свое недовольство в адрес разработчиков из Microsoft. Однако мы были не правы — такие визуальные средства имеются в VB 5 и VB 6 в виде автономного дополнения TabOrder (исходный код этого проекта находится в подкаталоге ...\MSDN98\98VS\1033\Samples\VB98\TabOrder\ для VB6 или в ...\Samples\CompTool\AddIns\TabOrder\).

Чтобы сделать Add-In (на примере VB 6.0), загрузите проект TabOrder.VBP и выполните команду File|Make TabOrder.DLL. В результате вы не только создадите исполняемый модуль, но и зарегистрируете новое дополнение, которое появится под названием Tab Order Sample Addin в окне Add-Ins|Add-In Manager среды Visual Basic. Для работы с TabOrder необходимо обычным образом установить режим загрузки в этом окне, и тогда на панели инструментов VB появится кнопка TabOrder Window для запуска этого дополнения (рис. 306-2).

Рис. 306-2

Примечание. *При работе в VB 5 для включения TabOrder (созданного в этой же среде!) в окно Add-In Manager необходимо добавить в секцию [Add-Ins32] файла VBADDIN.INI строку TabOrder.Connect=0.*

Теперь создайте в VB какой-нибудь тестовый пример с формой, на которой разместите несколько разных элементов управления, и щелкните кнопку TabOrder Window. В появившемся окне (рис. 306-3) можно корректировать последовательность перемещения по элементам управления формы.

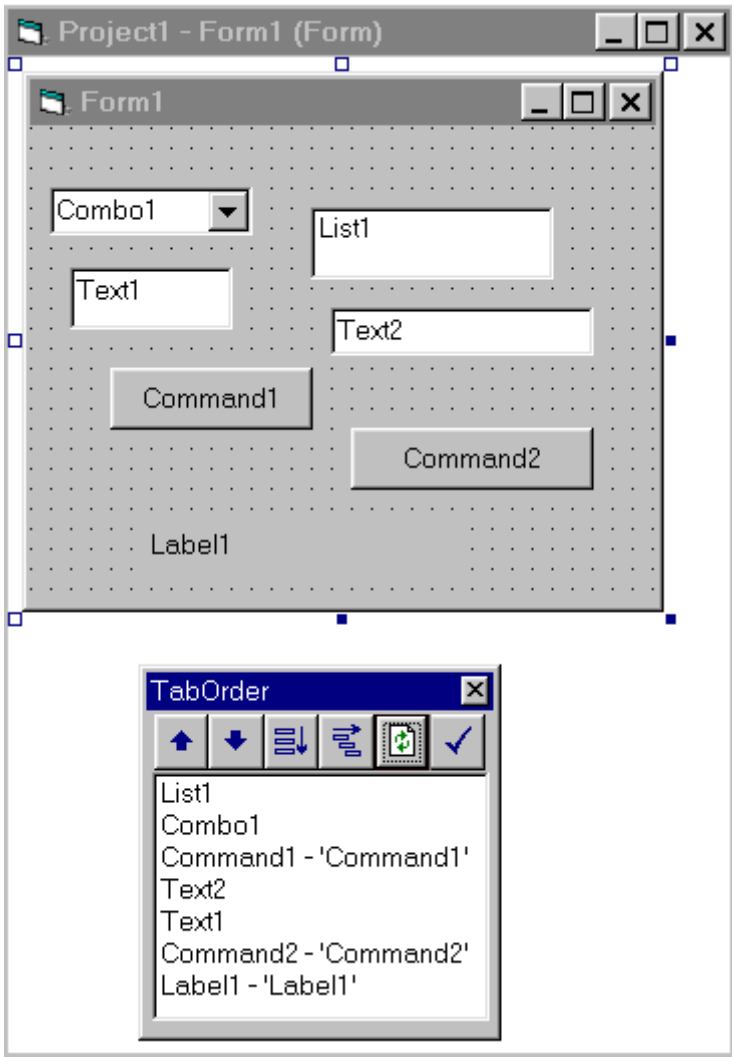


Рис. 306-3

Однако здесь имеются дополнительные функции, которых нет в аналогичной утилите MS Forms 2.0. Обратите внимание, что на нашей тестовой форме все элементы управления расположены на разных расстояниях от верхней и правой границ формы. С помощью соответствующих кнопок на панели инструментов можно упорядочить список по вертикальному (рис. 306-4) или горизонтальному (рис. 306-5) расположению визуальных компонентов. Кроме того, программист без перезагрузки этой утилиты может восстановить исходную последовательность списка.

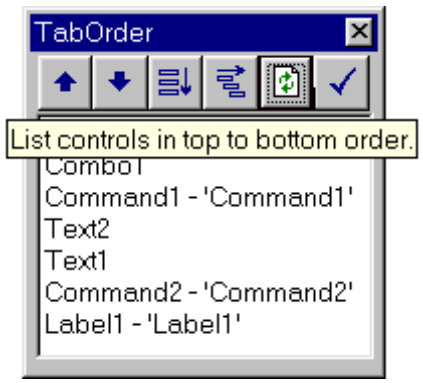


Рис. 306-4

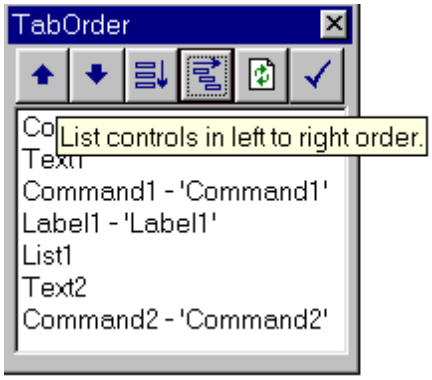


Рис. 306-5

Вообще, мы рекомендуем внимательно изучить проект TabOrder, так как он представляет собой интересный пример создания Add-In, использования User-документов и ресурсных файлов. Но один вопрос все же остается — зачем нужно было делать для VB-форм и MS Forms 2.0 две похожие, но тем не менее разные утилиты?

Совет 307. Как удалить каталог

VB содержит довольно много встроенных функций и операторов для управления файлами и каталогов. Совершенно очевидно, что эти средства работают через Win API, и понятно, что последние обладают более широкими возможностями. Чаще всего встроенные функции VB реализуют усеченный вариант соответствующей API-функции (например, оператор MkDir представляет собой частный случай функции CreatDirectory). Многие же возможности API вообще не представлены в виде VB-операторов. В частности, это относится к операции удаления каталогов, которая может быть выполнена таким образом:

```
Private Declare Function RemoveDirectory& Lib "kernel32" Alias  _
     "RemoveDirectoryA" (ByVal lpPathName As String)

  ' Удаление каталога (пустого!)
  PathName$ = "D:\t\"     ' в конце имени "\" необязательна
  '
  code& = RemoveDirectory(PathName)
  If code& = 0 Then
      ' операция удаления не была выполнена
  Else
      ' каталог удален
  End If
```

Здесь нужно обратить внимание на несколько моментов:

1. Удаляемый каталог должен быть пустым.
2. Имя каталога не обязательно должно заканчиваться на "\" — операция будет успешно выполнена с именем "D:\t".
3. Почему-то при наличии ошибки обращение к функции GetLastError выдает исключительно 0 (то есть нельзя уточнить характер ошибки).

Совет 308. Как удалить непустой каталог

Но, как удалить каталог, если в нем находятся файлы и вложенные подкаталоги? Для этого потребуется написать процедуру DeleteDir, которая будет перед удалением каталога просматривать и удалять его содержимое. Обращение к ней может выглядеть следующим образом:

```
Dim DirCount%, FileCount% ' число удаленных каталогов и файлов
DirName$ = "d:\t\"  ' удаляемый каталог
DirCount = 0: FileCount = 0    ' начальная установка
Result& = DeleteDir&(DirName$, DirCount%, FileCount%)
' Result = 0 - удален
```

А сама процедура будет содержать такой код:

```
Public Function DeleteDir&(DirName$, DirCount%, FileCount%)
  '
  ' Удаление каталога со всем его внутренним содержанием

  ' ВХОД:
  '     DirName$ — имя удаляемого каталога
  ' ВЫХОД:
  '    DeleteDir - код завершения (0 - удален, <>0 - не удален
  '    DirCount, FileCount — текущее число удаленных каталогов и файлов
  '    (перед первым обращении эти переменные должны быть обнулены)
  '
  Dim Result&
  '
  ' 1. Удалить все внутри каталога
  Call DeleteDirAll(DirName$, DirCount%, FileCount%, Result&)
  If Result = 0 Then ' без ошибок
    ' 2. удалить сам каталог
    Result = RemoveDirectory(DirName$)
    If Result = 0 Then
      Result = -1
    Else
      Result = 0: DirCount = DirCount + 1
    End If
  End If
  DeleteDir = Result ' код завершения
End Function
```

Но очевидно, что ключевой задачей здесь является написание процедуры DeleteDirAll, которая будет удалять внутреннее содержание каталога — файлы и подкаталоги. Тут не обойтись без использования рекурсии, и поэтому полезно вспомнить наши советы 229-231[51]. Более того, нам понадобится готовый код подпрограммы CurrentDirCounter ("как

[51] http://www.visual.2000.ru/develop/ms-vb/tips/0001-2.htm

есть"), а также продпрограмма FileNameCase, которую можно использовать в качестве заготовки и создать DeleteDirAll:

```
Public Sub DeleteDirAll(DirName$, DirCount%, FileCount%, Result&)
  '
  ' Удаление всех файлов и подкаталогов заданного каталога
  '
  ReDim arrFile$(100) ' для имен элементов оглавления
  Dim CountFile%, NewPathName$, i%
  '
  On Error GoTo ErrorDeleteFile
  ' формируем список файлов каталога
  Call CurrentDirCounter(DirName$, "*.*", CountFile%, arrFile$(), 0)
    ' удаление файлов
  If CountFile > 0 Then    ' есть файлы в подкаталоге
    For i = 1 To CountFile
      Kill DirName$ & arrFile$(i) ' удалить файл
      FileCount = FileCount + 1 ' счетчик удаленных файлов
    Next
  End If
  'формирование списка подкаталогов
  Call CurrentDirCounter(DirName$, "", CountFile%, arrFile$(), vbDirectory)
  If CountFile > 0 Then    ' есть подкаталоги
    For i = 1 To CountFile
      ' формирование имени вложенного подкаталога
      NewPathName$ = DirName$ + arrFile$(i) + "\"
      '!! рекурсивное обращение к функции удаления каталога
      ' с новым именем, но старыми значениями счетчиков
      Result = DeleteDir&(NewPathName$, DirCount%, FileCount%)
      If Result <> 0 Then Exit Sub ' была ошибка
    Next
  End If
Exit Sub
ErrorDeleteFile:
  Result = Err ' ошибка при удалении файла
End Sub
```

Андрей Колесов, Ольга Павлова

© 2000, Андрей Колесов, Ольга Павлова
Авторский вариант. Статья была опубликована с незначительной литературной правкой в "КомпьютерПресс" N 10/2000, CD-ROM.

Совет 309. Как получить сводку о файле MP3

Вполне вероятно, что, прежде чем начать проигрывать файл формата MP3 (это можно сделать, подключив средства Windows Media Player к VB-приложению), вы захотите узнать название песни, имя исполнителя и пр. Если MP3-файл использует наиболее популярное кодирование тэгов ID3, то сделать это очень легко. Данный стандарт записывает в последние 128 байтов сводную информацию, которую можно прочитать с помощью, например, такого кода:

```
Private Type TagInfo
  Tag As String * 3         ' признак ID3
  Songname As String * 30   ' название песни
  artist As String * 30     ' имя артиста
  album As String * 30      ' название альбома
```

```
  year As String * 4        ' год издания
  comment As String * 30    ' комментарий
  genre As String * 1       ' тип жанра
End Type

Dim FileName As String
Dim CurrentTag As TagInfo

Private Sub ReadMP3FileInfo(MP3FileName$)
  ' чтение сводки MP3-файла
  On Error Resume Next

  Open FileName For Binary As #1
  With CurrentTag
    Get #1, FileLen(FileName) - 127, CurrentTag
    Close #1
    If Not .Tag = "TAG" Then
        MsgBox "Это не стандарт ID3"
        Exit Sub
    End If
    ' перезапись в текстовые поля, удаляя пробелы справа
    txtTitle = RTrim(.Songname)
    txtArtist = RTrim(.artist)
    txtAlbum = RTrim(.album)
    txtYear = RTrim(.year)
    txtComment = RTrim(.comment)
    ' если есть список жанров, то можно установить индекс
    Combo1.ListIndex = .genre - 1
  End With
End Sub
```

Совет 310. Приоритетность использования оператора With

Применение оператора With позволяет не только сократить время на ввод программы и улучшить читабельность кода, но и повысить скорость выполнения приложения. Использовать With можно как для объектов, так и для переменных пользовательского типа (User-Defined type, UDT). Вопрос заключается в следующем: для какого типа данных лучше использовать With? Ответ: в первую очередь для объектов, так как в этом случае будет снижаться время доступа к их свойствам. Например, у вас имеется структура, в которой хранится набор параметров:

```
Private Type TBtnSettings
  BgColor As Long
  FontSize As Integer
End Type
Private mOrigSettings As TBtnSettings
```

Вам нужно присвоит эти параметры свойствам командной кнопки. Конструкция:

```
Private Sub Command1_Click()
  ' более быстрый вариант
  With Command1
    .BackColor = mOrigSettings.BgColor
    .FontSize = mOrigSettings.FontSize
  End With
End Sub
```

будет работать быстрее, чем, например, такая конструкция:

```
Private Sub Command1_Click()
  ' менее быстрый вариант
  With mOrigSettings
    Command1.BackColor = .BgColor
    Command1.FontSize = .FontSize
  End With
End Sub
```

Совет 311. Следите за состоянием батарей ноутбука

Элемент управления SysInfo, поставляемый с VB (в окне Components он называется Microsoft SysInfo Control Version 6.0 или 5.0), довольно редко применяется разработчиками. И напрасно, ведь он позволяет получать информацию о параметрах и событиях операционной системы, событиях plug-and-play и др. В данном случае мы покажем, как с его помощью можно прочитать сведения о состоянии батарей ноутбука.

Свойство ACStatus позволяет узнать, используется ли в данный момент батарея для питания:

```
Select Case SysInfo1.ACStatus
  Case 0
    MsgBox "Питание сейчас не от батареи"
  Case 1
    MsgBox "Питание сейчас не от батареи"
  Case 255
    MsgBox "Состояние батареи неизвестно"
End Select
```

Свойство BatteryLifePercent определяет процент зарядки батареи:

```
Dim PerCentLeft As String
If SysInfo1.BatteryLifePercent <> 255 Then
  PerCentLeft = SysInfo1.BatteryLifePercent
  MsgBox PerCentLeft & "%"
Else
  MsgBox "Состояние заряда батареи неизвестно"
End If
```

Если ноутбук питается сейчас от батареи, то полезно узнать, сколько времени еще можно работать с ней:

```
If SysInfo1.BatteryLifeTime <> &HFFFFFFFF Then
  MsgBox "Осталось работать" & _
  Format(TimeSerial(0, 0, SysInfo1.BatteryLifeTime), "h:mm")
Else
  MsgBox "Нельзя определить время работы батареи"
End If
```

Совет 312. Следите за правильным использованием типов данных

Мы уже не раз отмечали, что Visual Basic позволяет разработчику весьма вольное использование типов переменных, выполняя неявное преобразование. Например, такая конструкция в VB вполне допустима и даже выдает правильный результат:

```
Dim iValue As Integer
Text1.Text = "5"
iValue = 4 * Text1.Text
```

```
Print iValue    ' будет напечатано 20
```

Однако с точки зрения теории языков программирования такое смешение типов в одном выражении является грубой ошибкой. К сожалению, многие разработчики привыкли к подобному стилю VB и считают его очень удобным, поскольку в данном случае нетнеобходимости тратить время на использование специальных функций преобразования типов данных. Следуя такому подходу, можно написать следующий код:

```
Select Case Text1.Text
  Case 1 to 12
    MsgBox "Допустимая величина"
  Case Else
    MsgBox "Недопустимая величина"
End Select
```

При этом можно подумать, что при вводе в текстовое поле величины от 1 до 12 будет выдаваться сообщение "Допустимая величина". Однако на самом деле код будет работать иначе: "Допустимая величина" будет выдаваться только для значений 1, 10, 11 и 12. Для чисел от 2 до 9 пользователь получит сообщение "Недопустимая величина".

Трудно сказать, почему в случае арифметического выражения для строки "5" преобразование выполняется верно, а в операторе Select — неверно. Следует только иметь в виду, что при использовании неявного преобразования могут возникнуть подобные проблемы. Придерживаясь же классических принципов программирования и потратив немного времени на ввод нескольких дополнительных символов, вы легко избежите этих проблем. Такой вариант кода будет надежно работать для любых числовых значений текстового поля:

```
Select Case Val(Text1.Text)
  Case 1 to 12
    MsgBox "Допустимая величина"
  Case Else
    MsgBox "Недопустимая величина"
End Select
```

Совет 313. Как контролировать наличие кавычек в названиях

Наш читатель из Томска прислал такой вопрос:

> *"У меня в базе данных есть таблица "Поставщики". В ней имеется поле "Название", в котором названия фирм могут содержать кавычки, например <ООО "МММ">. Мне необходимо сформировать строку SELECT с критерием поиска WHERE [Название] = ..., но когда я делаю это, у меня возникает синтаксическая ошибка (лишние кавычки):*
> `WHERE [Название] = "ООО "МММ""`

Вопрос сформулирован таким образом, что не совсем понятно, как читатель формирует строку SELECT и зачем использует такой вариант поиска. Поэтому попробуем рассмотреть эту проблему с разных сторон.

1. Обработка символьных данных, внутри которых содержатся двойные кавычки, представляет некоторые трудности, так как этот символ используется в качестве скобок для написания символьных литералов. Решается эта проблема общепринятым способом: внутри литерала каждая кавычка записывается дважды. Поэтому при синтаксическом разборе текста исходного кода первая кавычка считается правой скобкой, далее каждая

пара заменяет одну внутри строки, непарная является левой скобкой. Соответственно, чтобы присвоить приведенной выше переменной название фирмы, нужно написать:

```
2.   CompanyName$ = "ООО ""МММ"""
```

Отметим, что такая запись не очень хорошо воспринимается визуально, поэтому при использовании большого числа подобных строк можно вместо двойных кавычек применять какой-то другой символ, а потом воспользоваться простой подпрограммой замены символов. Например, в VB6 это может выглядеть таким образом (при вводе мы используем одинарные кавычки):

```
CompanyName$ = Replace34("ООО 'МММ'")
...
Sub Replace34$ (Word$)
    Replace34$ =    Replace(Word$, Chr$(39), Chr$(34))
End Sub
```

3. С учетом проблемы с двойными кавычками многие языки допускают в качестве логических скобок для строковых литералов использование или двойных, или одинарных кавычек. В том числе и SQL-обращения. В нашем случае оператор должен выглядеть следующим образом (чтобы были видны кавычки, я использую имя <ООО "МММ" О>):

```
4.   SELECT ... Where Firm = 'ООО "МММ" О'
```

Соответственно обращение к набору данных будет выглядеть так:

```
Data1.RecordSource = "SELECT... = 'ООО ""МММ"" О'    "
```

Для иллюстрации проведем небольшой тестовый пример таблицы с такими записями названий фирм:

```
ООО "МММ"
ООО 'МММ'
ООО МММ
ОООМММ
```

Чтобы найти первую фирму, нужно сделать обращение типа:

```
Data1.RecordSource = "SELECT... = 'ООО ""МММ"" О' "
```

Чтобы найти вторую:

```
Data1.RecordSource = ""ООО 'МММ' О"" "
```

5. Немного сложнее обстоит дело, когда название искомой фирмы вводится в поле, то есть заранее неизвестно, содержит оно двойные или одинарные кавычки. В этом случае при формировании SQL-запроса нужно определить, какие кавычки содержатся в наименовании, и соответственно выбрать другие. Для этого можно предложить такой вариант:

```
6.   '    FindFirm$ — искомое название.
7.      If Insrt (FindFirm$, Crh$(34)" Then
8.          d$ = Chr$(39) ' название содержит двойные кавычки
9.      Else
10.         d$ = Chr$(34) ' название содержит двойные кавычки
11.     End If
12.     SQL$ = "Select ... Where Firm = " & d$ & FindFirm$ & d$
```

Очевидно, что в подобных запросах проблема возникает при наличии в названии кавычек двух типов. (Хотя, наверное, и тут можно как-то исхитриться.)

13. Вообще-то лучше осуществлять поиск так, чтобы он не зависел от кавычек в названии. Тут может пригодиться оператор Like (подробно его работа рассматривается в статье "Особенности работы со строковыми переменными в Visual Basic"[52], КомпьютерПресс N 1/2000).

    Например, для приведенной выше базы из четырех записей запрос:

    ```
    Firm Like 'OOO*MMM*'
    ```

    найдет все четыре записи, а запрос:

    ```
    Firm Like 'OOO *MMM*'
    ```

    найдет только первые три записи.

Совет 314. Вывод подсказок на панель StatusBar

Как известно, при передаче фокуса элементу управления на экран может выдаваться подсказка (с помощью свойства ToolTipText, которое есть у многих элементов управления). Иногда бывает полезным, чтобы такая подсказка появлялась в строке статуса. Это делается следующим образом:

```
Private Sub Text1_GotFocus()
  StatusBar1.Panels(1).Text = "Text1" ' вывести текст
End Sub

Private Sub Text1_LostFocus()
  StatusBar1.Panels(1).Text = ""      ' очистить
End Sub
```

Совет 315. Программное управление раскладкой клавиатуры

Обычно переключение раскладки клавиатуры с русского языка на английский и наоборот выполняется при помощи комбинации "горячих" клавиш. Но иногда удобнее для установки каждой раскладки использовать определенную комбинацию. Это тем более полезно при работе с многоязычными документами (поддержка трех языков в странах бывшего СССР — дело обычное). Короче говоря, вопрос упирается в то, как программно устанавливать нужную раскладку клавиатуры. При работе с пакетами MS Office это делается очень просто:

```
Application.Keyboard (LangID)
```

Однако такую установку можно провести только для списка раскладок клавиатуры, определенного в панели инструментов. Здесь нужно иметь в виду, что код региональной установки может не соответствовать значению параметра региона. Например, для моего компьютера код для русской раскладки &h04190419, то есть региональный код дублируется в старшей и младшей частях числа.

[52] http://www.visual.2000.ru/develop/ms-vb/cp9910/strvb-2.htm

Уточнить конкретный код раскладки можно очень просто: установить нужный режим, а потом определить его числовое выражение:

```
LangIDCurrentKeyBoard = Application.Keyboard
```

При создании приложения с помощью обычного VB подобные операции (но с более широкими возможностями) выполняются путем использования функций Win API.

Совет 316. Простые алгоритмы шифрования данных

Существует немало алгоритмов шифрования-дешифрования информации. И, наверное, еще много методов будет придумано в будущем. Мы же предлагаем использовать для подобных задач достаточно простой вариант на основе известной логической операции XOR, принцип которой описывается следующими формулами:

```
<Шифрованный код> = <Исходный код> XOR <Шифр>
<Исходный код> = <Шифрованный код> XOR <Шифр>
```

Таким образом, кодирование и последующее декодирование информации выполняются с помощью одного числового кода (шифра) и операции XOR. В программной реализации это выглядит следующим образом:

```
SourceArray() As Byte ' байтовый массив с исходной информацией
KeyWord() As Byt      ' байтовый массив с ключом-шифром
ResultArray() As Byte ' байтовый массив с зашифрованной информацией
```

Шифрование и дешифрирование данных выполняются с помощью одной и той же процедуры:

```
' получаем зашифрованный массив
ResultArray = EncryptDecrypt(SourceArray, KeyWord)
' записываем шифрованный массив и в нужный момент восстанавливаем
SourceArray = EncryptDecrypt(ResultArray, KeyWord)
```

Процедура шифрования выглядит следующим образом:

```
Public Function EncryptDecrypt _
   (Source() As Byte, KeyWord() As Byte) As Byte()
  '
  ' кодирование-декодирование массива
  ReDim Result(LBound(Source) To UBound(Source)) As Byte
  Dim k&, klw&, kuw&, k1&
  klw = LBound(KeyWord):
  kuw = UBound(KeyWord): k1 = kuw
  For k = LBound(Source) To UBound(Source)
    k1 = k1 + 1: If k1 > kuw Then k1 = klw
    Result(k) = Source(k) Xor KeyWord(k1)
  Next
  EncryptDecrypt = Result()
End Function
```

Здесь следует сделать несколько замечаний.

1. Если вы кодируете информацию, то имеет смысл следить за целостностью данных. Ведь "враг", хотя и не поймет содержание данных, но навредить может, испортив их (а вы и не

заметите этого!). Эта проблема решается, например, вычислением контрольной циклической суммы:

```
  Dim SourceCycle(0 To 3) As Byte
  Call CycleSumma(SourceArray, SourceCycle)

  Public Sub CycleSumma(Source() As Byte, Cycle() As Byte)
    ' вычисление циклической суммы
    Dim k&, nc%
    For k = LBound(Source) To UBound(Source)
      nc = k Mod 4
      Cycle(nc) = Cycle(nc) Xor Source(k)
    Next
  End Sub
```

13. Следует иметь в виду, что приведенный здесь алгоритм будет хорошо работать для достаточно неоднородного массива исходных данных (с низким значением эндотропии). Однако если массив состоит из одних нулей, то шифрованный будет состоять из легко интерпретируемой последовательности кодового массива. То есть такой алгоритм плохо подходит для файлов MS Office, в которых имеются огромные пустоты, заполненные байтами с тривиальными нулями или двоичными единицами. Но для обычных текстовых данных и тем более — для архивных файлов, где эндотропия практически равна единице, он будет просто идеальным.
14. Нужно быть очень внимательным при использовании символьных данных ключа и исходного массива. Например, если у вас есть такие исходные данные:

```
  strSource$ = "Исходный текст, который мы хотим закодировать"
  strKey$="Шифр"
```

но преобразование строки в байтовый массив выглядит очень неудачными:

```
SourceArray() = strSource$
KeyWord()= strKey$
```

Дело в том, что в этом случае мы получаем кодировку Unicode, где каждый нечетный байт будет иметь фиксированное значение (появляется однородность массива, которую лучше избегать). В этом случае лучше работать с использованием ANSI-кода:

```
SourceArray() = StrConv(strSource$, vbFromUnicode)
KeyWord()= StrConv(strKey$, vbFromUnicode)
```

17. Очевидно, что вы можете повышать уровень кодирования простым увеличением числа байтов в кодовом массиве, не принимая во внимание ограничения ФАПСИ на длину ключа. Например, кодом может быть достаточно длинная текстовая фраза: "Дети в школу собирайтесь, петушок пропел давно".

Совет 317. Как вывести информацию в элементе управления WebBrowser, не пользуясь HTML-файлом

При использовании элемента управления WebBrowser в VB можно вывести информацию таким образом, чтобы конечный пользователь не имел к ней доступа. Для этого следует просто вставить HTML-код непосредственно в элемент управления.

Продемонстрируем данный метод на следующем примере. Вначале создадим пустой документ внутри элемента управления WebBrowser, а затем введем туда HTML-текст, не используя никакого внешнего HTML-файла. Таким образом мы не только упрощаем

работу, но и защищаем свой HTML-код: если пользователь выберет команду ViewSource для просмотра кода, то все, что он увидит, будет <HTML></HTML>.

```
Option Explicit
Property Set Doc(Document As Object)
    Set CurrentDoc = Document
End Property

Private Sub Form_Load()
  Dim strHTMLText As String
  ' Создаем пустой документ в
  ' элементе управления WebBrowser
  WebBrowser1.Navigate2 "about:Blank"
  ' Web-браузеру может понадобиться некоторое
  ' время для обработки каждой команды
  DoEvents
  On Error GoTo WaitAwhileLonger
  ' Устанавливаем цвет фона документа
  WebBrowser1.Document.body.bgcolor = "#000000"
  ' Задаем HTML-текст с помощью кода
  ' или информации из базы данных
  strHTMLText = "<html>" & vbCrLf & "<head>" & _
    vbCrLf & "<title>Наш проект</title>" & _
    vbCrLf & "</head>" & vbCrLf & _
    "<body><p align=""center""> " & _
    "<font face=""Arial"" size=""5"" " & _
    "color=""#FFFFFF""><strong> " & _
    "Советы по VB & VBA</strong></font> " & _
    "</p><p align=""center""> " & _
    "<a href=""http://www.visual.2000.ru""> " & _
    "Посетите наш сайт</a></p></body>" & _
    vbCrLf & "</html>"
  strHTMLText = strHTMLText & "<head>" & vbCrLf
  ' Отправляем HTML-текст непосредственно
  ' в элемент управления WebBrowser
  WebBrowser1.Document.body.innerhtml = strHTMLText
  Exit Sub
  '
WaitAwhileLonger:
  Debug.Print Hex(Err.Number), Err.Description
  DoEvents
  Resume
End Sub
```

Элементу управления WebBrowser иногда требуется некоторая "поддержка", чтобы полностью закончить выполнение задания, прежде чем перейти к следующему. Поэтому мы используем здесь ловушку ошибок, которая позволяет компоненту WebBrowser "перевести дух", а затем вновь вернуться к работе.

Совет 318. Как преобразовать OLE_COLOR в фактическое значение функции RGB

Вы когда-нибудь пытались передать системную VB-константу, задающую цвет (например, vbButtonFace), в API-функцию? На практике часто бывает необходимо использовать системные цвета при вызове GDI-интерфейса (Graphical Device Interface — Интерфейс графических устройств), а вы предпочитаете работать с системными цветовыми VB-константами. Проблема заключается в том, что GDI-интерфейс не знает, что делать с такими константами, и в результате вы всегда получаете черный цвет.

Мы предлагаем вам следующее решение. Используйте API-функцию OleTranslateColor, которая читает любую из таких констант, а затем преобразует ее в буквенные RGB-цвета, которые понятны для GDI-интерфейса. Это можно сделать так:

```
Option Explicit

Private Declare Function OleTranslateColor _
  Lib "oleaut32.dll" (ByVal lOleColor As Long, _
  ByVal lHPalette As Long, lColorRef As _
  Long) As Long

Public Function TranslateColor(inCol As _
  OLE_COLOR) As Long
  '
  Dim retCol As Long
  OleTranslateColor inCol, 0&, retCol
  TranslateColor = retCol
End Function
```

Теперь просто вызовите функцию TranslateColor, используя конкретную системную цветовую константу, для получения необходимого вам цвета, например так:

```
Private Sub Form_Load()
  Dim newColor As Long
  newColor = TranslateColor(&H80000001)
  Form1.BackColor = newColor
End Sub
```

Обратите внимание, что если вы передаете в функцию TranslateColor стандартное значение функции RGB, то оно возвращается в неизменном виде. Поэтому нет смысла беспокоиться о том, в каком виде хранить значение цвета — как системную константу или как фактическое значение цвета.

Совет 319. Один из способов создания уникального строкового идентификатора

Если существует потребность в уникальном строковом идентификаторе, а у вас нет возможности проверить, является ли сгенерированный вами идентификатор уникальным, воспользуйтесь идентификатором Universally Unique ID (UUIID) или Globally Unique ID (GUID). UUID представляет собой 128-битовое число, которое генерируется на базе текущего времени и сетевой интерфейсной платы (Network Interface Card — NIC) вашего компьютера. Это гарантирует, что получаемая строка будет уникальной (по крайней мере, в рамках вашей сети и до наступления 3400 года).

Следующая функция создает идентификатор UUID и преобразовывает его в 36-байтовые строки. Для этого в текст модуля вставьте такой код:

```
Option Explicit

Private Declare Function UuidCreate Lib _
    "rpcrt4.dll" (pId As UUID) As Long
Private Declare Function UuidToString Lib _
    "rpcrt4.dll" Alias "UuidToStringA" _
    (uuidID As UUID, ppUuid As Long) As Long
Private Declare Function RpcStringFree Lib _
    "rpcrt4.dll" Alias "RpcStringFreeA" _
    (ppStringUiid As Long) As Long
```

```
Private Declare Function CopyMemory Lib _
    "kernel32.dll" Alias "RtlMoveMemory" _
    (pDst As Any, pSrc As Any, ByVal nSize _
    As Long) As Long

Private Type UUID
  Data1 As Long
  Data2 As Long
  Data3 As Long
  Data4(8) As Byte
End Type

Public Function GenUuid(sUuid As String) As Boolean
  Const RPC_S_OK As Long = 0
  Const SZ_UUID_LEN As Long = 36
  Dim uuidID As UUID
  Dim sUid As String
  Dim ppUuid As Long
  '
  sUid = String(SZ_UUID_LEN, 0)
  If UuidCreate(uuidID) = RPC_S_OK Then
    If UuidToString(uuidID, ppUuid) = _
      RPC_S_OK Then
      CopyMemory ByVal sUid, ByVal ppUuid, _
          SZ_UUID_LEN
      If RpcStringFree(ppUuid) = RPC_S_OK Then
        sUuid = sUid
        GenUuid = True
      End If
    End If
  End If
End Function
```

Использовать эту функцию можно, например, так:

```
Dim sId As String
Call GenUuid(sId)
MsgBox "Идентификатор = " & sId
```

Несомненно, вы найдете лучшее применение созданным таким образом идентификаторам, чем просто выводить их на экран. Помните, однако, что существует потенциальная возможность с помощью подобных идентификаторов однозначно определить, на какой машине они были сгенерированы. А это уже связано с вопросами безопасности, хотя, возможно, они не имеют для вас никакого значения.

Совет 320. Как с помощью метода OpenSchema получить информацию о структуре базы данных

Метод OpenSchema объекта Connection позволяет получить информацию о структуре базы данных от провайдера. Иными словами, он дает возможность просматривать коллекции данных, хранящиеся в базе данных OLE DB, не составляя перечня самих коллекций. Этот метод возвращает набор данных с полями, описывающими членов коллекции. Так, если использовать метод OpenSchema с запросом adSchemaTables и провайдером данных Jet 4.0, мы получим информацию о локальных таблицах, связанных таблицах, передаваемых запросах, системных таблицах и таблице Access.

В вашем распоряжении свыше 30 типов запросов, с помощью которых можно получить информацию о содержимом источника данных OLE DB. Помимо этого запрос

adSchemaTables возвращает информацию о таких знакомых объектах баз данных, как индексы, основные ключи, внешние ключи, процедуры и разрезы данных.

Следующий пример демонстрирует, как с помощью метода OpenSchema и провайдера Jet 4.0 получить список разрезов данных:

```
Option Explicit

Public Sub OpenSchemaX()
  Dim cnn1 As New ADODB.Connection
  Dim rstSchema As ADODB.Recordset
  cnn1.Open "Provider=Microsoft.Jet.OLEDB.4.0; _
      Data Source=C:\VB-DB\Nwind.mdb;"
  Set rstSchema = cnn1.OpenSchema(adSchemaTables)
  ' Просмотр только разрезов данных; другие
  ' критерии выбора включают TABLE, LINK,
  ' PASS-THROUGH, ACCESS, TABLE и SYSTEM TABLE
  Do Until rstSchema.EOF
    If rstSchema.Fields("TABLE_TYPE") = "VIEW" Then
      MsgBox "View name: " & rstSchema.Fields _
         ("TABLE_NAME") & vbCr
    End If
    rstSchema.MoveNext
  Loop
  rstSchema.Close
  cnn1.Close
End Sub

Private Sub Form_Load()
  Call OpenSchemaX
End Sub
```

Совет 321. Как обнаружить неиспользуемые объекты

Неправильное использование объектов или ссылок на объекты может привести к тому, что они останутся в оперативной памяти, снижая тем самым быстродействие приложения. Чтобы избежать подобных ситуаций и найти "плохой код" в своей программе, можно применять системные утилиты и инструменты третьих фирм. Однако это не самый простой путь. Мы рекомендуем воспользоваться технологией обработки событий, имеющейся в VB5/VB6.

Создайте новый проект ActiveX DLL, содержащий один класс ResidentObjs.cls. Опишите некоторое событие в разделе Declarations для этого класса. Затем добавьте метод, который запускает данное событие, и назовите полученную подпрограмму DetectAllObjects:

```
Option Explicit
Public Event ObjectNotification()

Public Sub DetectAllObjects()
  DoEvents
  RaiseEvent ObjectNotification
End Sub
```

Создайте библиотеку командой File|Make ListObjects.dll.

Теперь в своем приложении добавьте ссылку к только что созданной библиотеке, а в BAS-модуле опишите глобальную переменную как ResidentObjs. Затем напишите код, который будет создавать экземпляр этой переменной при инициализации приложения:

```
Sub Main()
  Set oListObject = New ListObjects.ResidentObjs
End Sub
```

В каждом классе своего проекта опишите переменную ResidentObjs с помощью ключевого слова WithEvents. Потом установите эту объектную переменную как глобальную переменную для инициализации класса или другой функции класса, которая вызывается каждый раз при создании объекта данного класса. В функции события можно написать любой код, который бы сообщал о текущем экземпляре объекта. Например, вы можете добавить Debug.Print или какой-либо другой код, записывающий имя класса в файл.

Теперь вы можете вызвать подпрограмму DetectAllObjects из любого места кода приложения. При этом количество вызовов функции соответствует количеству объектов, находящихся в оперативной памяти в текущий момент времени.

Как же работает описываемый здесь механизм? События — это разновидность анонимных коммуникационных сообщений, которые отправляются каждому экземпляру объекта в приложении. Благодаря этому с помощью предложенного нами достаточно простого кода можно легко обнаружить недопустимые объекты.

Андрей Колесов, Ольга Павлова

© 2000, Андрей Колесов, Ольга Павлова
Авторский вариант. Статья была опубликована с незначительной литературной правкой в журнале "КомпьютерПресс" 11/2000, компакт-диск.

Совет 322. Получение даты последней коррекции файла

Для работы иногда полезно получить данные о дате создания или последней коррекции файла. Для этого можно использовать функцию FileDateTime(ИмяФайла), например:

```
Dim MyDateTime As Date
MyDateTime = FileDateTime("D:\Calc.bas")
```

С помощью этой же функции можно определить время создания каталога, но следует иметь в виду, что имя каталога в данном случае не должно заканчиваться на обратную косую черту:

```
MyDateTime = FileDateTime("D:\TMP\")      ' имя каталога задано неверно
MyDateTime = FileDateTime("D:\TMP")       '    имя каталога задано правильно
```

Но получить дату создания корневого каталога (то есть диска) с использование FileDateTime нельзя.

Совет 323. Преобразование кода цвета из DOS в Windows

Как известно, в стандартном варианте DOS использовалось 16 цветов, а сейчас в Windows их 16 777 216 (16M). Точнее говоря, такая палитра определяется не операционной системой, а техническими характеристиками мониторов.

Несмотря на такое мощное расширение состава палитры, для решения многих задач бывает удобнее воспользоваться ограниченным количеством цветов. К тому же порой необходимо точно воспроизвести в VB-цвета, которые использовались в DOS (например, при работе с QuickBasic).

В любом случае, возникает задача преобразования кодов цветов из DOS в Windows, которая, казалось бы, легко решается с помощью встроенной функции QBColor:

```
WindowsColor = QBColor(DosColor)
```

Однако небольшое исследование показывает, что такой вариант представляет собой весьма приблизительное решение. С помощью следующей конструкции:

```
For i = 0 To 16
  Debug.Print i; QBColor(i)
Next
```

можно получить таблицу соответствия кодов DOS и Windows[53].

Здесь хорошо видны принципы кодирования цветов в DOS и Windows. В обоих случаях цвет задается комбинацией красного, зеленого и синего (Red, Green, Blue). Но в первом варианте используется лишь один разряд (то есть две градации цвета — "есть" или "нет" данной составляющей), а во втором — целый байт (256 градаций). В DOS увеличение цветовой гаммы в два раза достигается наличием четвертого разряда, который задает нормальную или повышенную яркость (но не для каждого компонента, а для всей комбинации).

Казалось бы, на этом можно поставить точку, но из таблицы видно некоторое несоответствие в логике преобразования кода. Почти для всех цветов замена кода выполняется по такому правилу: для нормальной яркости двоичный разряд со значением 1 заменяется на &h80, для повышенной яркости — на &hFF. Но имеется также исключение из этого правила для цветов 7 и 8. Следует обратить внимание и на разные названия, например одинаковых цветов, которые используются в английском и русском вариантах Help. Оказывается, это не неточность перевода, а отражение реального несоответствия цветов в DOS и в Windows после преобразования кодов с помощью QBColor.

Визуальное сравнение изображений палитры из 16 цветов, полученных в среде QB и VB, показало заметное расхождение в красках, поэтому мы выполнили следующий эксперимент:импортировали графический файл с палитрой QB в VB и прочитали RGB-код каждого цвета. По результатам замеров мы сделали свою функцию преобразования кодов цветов из DOS в Windows:

```
Public Function QBColorMy(DosColor%) As Long
  ' Преобразование    кода из QB (16 цветов) в VB (RGB - 16M)
  ' Значения RGB-кодов получены на основе исследования
```

[53] http://www.visual.2000.ru/develop/ms-vb/tips/0011add.htm

```
  ' 16-цветной палитры, полученной в QB/DOS
  Select Case DosColor
    Case 0: QBColorMy = &H0         '    Black
    Case 1: QBColorMy = &HAA0000    ' Blue
    Case 2: QBColorMy = &HAA00&     '    Green
    Case 3: QBColorMy = &HAAAA00    ' Cyan
    Case 4: QBColorMy = &HAA        ' Red
    Case 5: QBColorMy = &HAA00AA    ' Magenta
    Case 6: QBColorMy = &H55AA&     '    Brown (! нарушение    алгоритма)
    Case 7: QBColorMy = &HAAAAAAA   '    White
    Case 8: QBColorMy = &H555555    ' Gray
    Case 9: QBColorMy = &HFF5555    ' Light Blue
    Case 10: QBColorMy = &H55FF55   '    Light Green
    Case 11: QBColorMy = &HFFFF55   '    Light Cyan
    Case 12: QBColorMy = &H5555FF   '    Light Red
    Case 13: QBColorMy = &HFF55FF   '    Light Magenta
    Case 14: QBColorMy = &H55FFFF   '    Yellow
    Case 15: QBColorMy = &HFFFFFF   '    Bright White
  End Select
End Function
```

Здесь следует обратить внимание на следующие моменты.

Для нормальной яркости двоичный разряд со значением 1 меняется на &hAA, а для повышенной яркости — на &hFF. Но в последнем случае нулевой двоичный разряд меняется на шестнадцатиричное 55 (в двоичном варианте это выглядит как 01010101).

Как ни странно, код 6 опять отличается от ожидаемой величины (было бы логично увидеть &hAAAA). При этом нужно отметить, что этот код действительно Brown (коричневый) — именно так он называется в QB. В документации VB он именуется как Yellow, что явно не соответствует действительности.

Совет 324. Как отличить обычные элементы управления от компонентов массива

Для того чтобы найти программным образом на форме элемента управления определенного типа, например TextBox, можно воспользоваться следующим кодом:

```
Dim ctl As Control
For Each ctl In Controls
  If TypeName(ctl) = "TextBox" Then
    MsgBox ctl.Name & " - это элемент управления   TextBox."
  End If
Next ctl
```

Но данный вариант не может отличить обычные элементы управления от массивов. Для решения этой проблемы следует воспользоваться такой конструкцией: TypeName(Controls(ctl.Name))

Для обычных элеменов управления она будет выдавать "TextBox", а для компонентов массива — "Object". Программа, которая будет выдавать более точную информацию, может выглядеть следующим образом:

```
Dim ctl As Control
  For Each ctl In Controls
  ' MsgBox TypeName(ctl)
  If TypeName(ctl) = "TextBox" Then
```

```
    If TypeName(Controls(ctl.Name)) = "TextBox" Then
      MsgBox ctl.Name & " - обычный    TextBox"
    Else
      MsgBox "Элемент    массива"    & ctl.Name & _
         "/ Индекс    = " & ctl.Index
    End If
  End If
Next ctl
```

Совет 325. Преобразование числа в двоичное представление и наоборот

В VB (и QB) имеются функции для представления целого числа в символьном виде в восьмеричной (Oct) и шестнадцатеричной (Hex) системах счисления:

```
Print Hex(59)  ' будет напечатано 3B
Print Oct(59)  ' будет напечатано 73
```

Соответственно можно также использовать специальный вид констант для задания чисел в восьмеричном или шестнадцатеричном виде:

```
iValue = &H3B   ' = 59 (приставка &h)
iValue = &O73   ' = 59 (приставка &o — второй символ буква "o")
```

Еще раз хотим обратить внимание на нюанс, связанный с преобразованием типов Integer в Long и наоборот. Проблема заключается в том, что мы имеем дело со знаковыми числами, в которых старший, знаковый разряд должен обрабатываться особым образом. Покажем эту ситуацию на примере шестнадцатеричной константы.

Правило формирования константы &H таково: если в ней указано поле приставки менее пяти символов, то формируется переменная Integer, в противном случае — Long. Если необходимо сразу получить тип Long, то константа должна иметь в конце суффикс &.

Соответственно, вы можете получить такой неожиданный результат:

```
Const MyConst As Long = &hFFFF
Print Hex$(MyConst)      ' будет напечатано FFFFFFFF
```

Это объясняется тем, что &hFFFF соответствует значению -1 (Integer). При переводе этого числа в тип Long получается код &hFFFFFFFF. Если же требуется сформировать число типа Long, в котором два старшие байта были нулевыми, то следует указать суффикс &:

```
Const MyConst As Long = &hFFFF&
Print Hex$(MyConst)      ' будет    напечатано FFFF
```

Если число задано в виде символьной записи, то его можно преобразовать следующим образом:

```
MyStr$ = "3B"
MyValue& = Val ("&h" & MyStr$)
```

Однако довольно часто бывает полезно представлять числа в двоичной системе счисления. Хотя шестнадцатеричный и восьмеричный вид чисел хорошо показывать двоичную структуру числа, все же такое преобразование требует внимания и не гарантирует от ошибок. Даже опытному программисту может понадобиться некоторое

время, чтобы записать число, в котором 7-й, 8-й и 10-й разряды (считая справа! — 1011000000) были бы равны единице, а остальные — нулю.

Для решения этой задачи можно использовать функции iBin% и sBin$, приведенные на листинге 325[54]:

```
Print Hex(iBin%("1011000000")  ' будет напечатано 2C0
Print sBin$(34)                ' будет напечатано   100010
```

Здесь нужно обратить внимание на то, что значительная часть кода связана с особой обработкой знакового разряда. Мы реализовали эти функции для целой типа Integer, учитывая два обстоятельства.

Именно такой диапазон чисел чаще всего используется для представления в двоичный формат (16 разрядов — это довольно много для визуального восприятия).

Для Long необходимо написать такие функции, чтобы они же по умолчанию работали и для Integer. Все это — из-за автоматического преобразования типов данных при обращении к процедурам и проблем такого преобразования, описанных выше. В этой связи приведем такой пример: строка "1000000000000000" (15 нулей справа) в терминах Integer равна -32768, а Long — +32768. Таким образом, для двух типов данных необходимы разные процедуры. (В VB 7 обещано появление Overloading — возможности использования одного и того же идентификатора для обозначения разных процедур. Выбор нужной процедуры зависит от типа аргумента.)

Совет 326. Функции с двухсторонним обращением

Довольно часто встречаются случаи парных процедур, которые выполняют связанные как бы прямые и обратные операции. Для удобства работы с ними полезно обозначить один из них идентификатором, при этом конкретная операция должна определяться формой записи:

```
y = MyFunc(x)
MyFunc(y) = x
```

Примером такого решения является популярная функция Mid:

```
Sym$ = Mid$(MyStr$, i, k)    ' выбирает    из строки фрагмент
Mid$(MyStr$, i, k) = Sym$    ' вставляет    фрагмент строку
```

Обратите внимание, что в первом случае конструкция называется функцией, а во втором — оператором (Statement). Если бы мы писали второй вариант сами, то данная процедура имела бы тип Sub, а ее запись выглядела бы так:

```
Call MyMid(MyStr$, i, k, Sym$)
```

Однако если использовать процедуры типа Property, то можно создать пару таких связанных функций одним именем. Например, мы хотим сделать две процедуры, одна из которых должна выбирать младший байт из длинной переменной, а другая — вставить его обратно:

[54] http://www.visual.2000.ru/develop/ms-vb/tips/0011add.htm

```
Public Property Get LowByte(LongVar As Long) As Byte
  ' прочитать младший    байт переменной LongVar
  LowByte = LongVar And &HFF
End Property

Public Property Let LowByte(LongVar As Long, ByVal NewByte As Byte)
  ' Записать в    младший байт переменной LongVar новое значение NewByte
  LongVar = (LongVar And &HFFFFFF00)    Or NewByte
End Property
```

Обращение к ним будет выглядеть следующим образом:

```
Dim LongVar$, ByteVar As Byte
LongVar = &h12345678
ByteVar = LowByte(LongVar)
Print Hex(ByteVar)    ' напечатано    78

ByteVar = &hF1
LowByte(LongVar) = ByteVar
Print Hex(LongVar)    ' напечатано    123456F1
```

На самом деле такие связанные функции представляют собой созданные выше iBin% и sBin$ (см. Совет 325[55]). Их можно легко преобразовать в пару процедур Property:

```
Public Property Get MyBin(Source%) As String
  ' Число Source%    в символьную запись двоичного кода
  MyBin = sBin$(Source%)
End Property

Public Property Let MyBin(Source%, ByVal NewStr As String)
  ' Символьную    запись двоичного кода NewStr$ в число Source%
  Source% = iBin%(NewStr)
End Property
```

В этом случае обращение к ним будет выглядеть следующим образом:

```
MyBin(intBin) = "1011000"
Print Hex(intBin)    ' напечатано 58
StrBin$ = MyBin(intBin)
MsgBox (StrBin)      ' напечатано 101100
```

Совет 327. Преобразование величины размера памяти

В составе Internet Explorer имеется библиотека SHLWAPI.DLL (Shell Windowing API), в которую включена функция, преобразующая число байтов в символьную строку типа "1,41 KB" или "1,32 MB". Обращение к этой функции выглядит следующим образом:

```
Private Declare Function StrFormatByteSize _
  Lib "shlwapi" Alias "StrFormatByteSizeA"    _
  (ByVal dw As Long, ByVal pszBuf As String, _
  ByVal cchbuf As Long) As Long

Public Function FormatKB(ByVal Amount As Long) As String
  Dim Buffer As String, i%
  Buffer = Space$(255) ' резервируем буфер
  Call StrFormatByteSize(Amount, Buffer, Len(Buffer))
  i = InStr(Buffer, vbNullChar)
```

[55] http://www.visual.2000.ru/develop/ms-vb/tips/0011.htm

```
  If i > 1 Then FormatKB = Left$(Buffer, i - 1)
End Function
```

Совет 328. Переключение вида заголовков элемента управления ListView

С помощью приведенного ниже кода можно переключать представление заголовков элемента управления ListView из плоского вида в объемное и наоборот:

```
Private Declare Function GetWindowLong _
   Lib "user32" Alias "GetWindowLongA" _
     (ByVal hwnd As Long, ByVal nIndex As Long) As Long
Private Declare Function SetWindowLong _
  Lib "user32" Alias "SetWindowLongA" _
  (ByVal hwnd As Long, ByVal nIndex As Long, _
  ByVal dwNewLong As Long) As Long
Private Declare Function SendMessage _
  Lib "user32" Alias "SendMessageA" _
  (ByVal hwnd As Long, ByVal wMsg As Long, _
  ByVal wParam As Long, lParam As Any) As Long

Private Const GWL_STYLE = -16
Private Const LVM_FIRST = &H1000
Private Const LVM_GETHEADER = LVM_FIRST + 31
Private Const HDS_BUTTONS = &H2

Private Sub Command1_Click()
  Call ToggleHeader(ListView1.hwnd)
End Sub

Public Sub ToggleHeader(lsvhWnd As Long)
  Dim hHeader As Long, lStyle As Long
  hHeader = SendMessage(lsvhWnd, LVM_GETHEADER, 0, ByVal 0&)
  lStyle = GetWindowLong(hHeader, GWL_STYLE)
  SetWindowLong hHeader, GWL_STYLE, lStyle Xor HDS_BUTTONS
End Sub
```

Совет 329. Генерация динамических HTML-страниц

Полтора года назад мы опубликовали (КомпьютерПресс N 5/99) [две статьи, посвященные созданию Web- и DHTML-приложений](#)[56]. Нам удалось создать вполне работоспособные примеры, однако технология разработки показалась нам весьма запутанной. Возможно, создавать динамические HTML-страницы проще с помощью знакомой ASP-технологии? С помощью VB это делается достаточно просто. Ниже приведены содержание ASP-страницы, которое можно сделать с помощью самого простого текстового редактора, и код процедуры, которая используется для генерации HTML-кода.

Тестовая Active Server Page:

```
<html>
<head></head>
<body>
<h2>Далее будет приведена некоторая информация пользователя </h2>
<%
Dim objRSServer
Set objRSServer = Server.CreateObject("TestASP.Htmlcrt")
objRSServer.WriteSomeThing Response, "Привет    всем!"
```

[56] http://www.visual.2000.ru/develop/ms-vb/cp9905-2/vb-web1.htm

```
%>
</body>
</html>
```

Программный код (проект TestASP.DLL, модуль класса Htmlcrt):

```
Public Sub WriteSomeThing _
   (ASPResponse As ASPTypeLibrary.Response, SomeText$)
   With ASPResponse
     .Write "<font size=4 color='#cc3366'>"
     .Write SomeText$
     .Write "</font>"
   End With
End Sub
```

В данном случае все выглядит достаточно понятным. Сначала мы создали ActiveX DLL с именем TestASP, в модуле класса Htmlcrt которой написали процедуру WhiteSomeThing (она превратилась, таким образом, в метод объекта TestASP.Htmlcrl). При этом для нашей DLL мы сделали ссылку на библиотеку Microsoft Active Server Page Object Library, чтобы получить доступ к объекту Response. Далее мы написали ASP-страницу с тривиальным скриптом — сначала создается объект TestASP.Htmlcrt, а затем следует обращение к его методу WhiteSomeThing с передачей объекта Response и некоторого набора параметров. В данном случае передается текстовая строка, которая выводится в HTML-страницу.

Напомним, что обращение к ASP-странице должно выполняться через Internet Infomation Server (NT) или Personal Web Server (Windows 9x). В последнем случае ASP-файл должен быть помещен в одну из папок сервера, обращение к которому из браузера выполняется по адресу http://localhost/MyDir/Test.ASP. Понятно, что никакого практического применения подпрограмма WhiteSomeThing не имеет, поэтому предлагаем вашему вниманию процедуру ShowRecordSetASP (листинг 329[57]), которая выводит на HTML-страницу содержимое заданного набора данных. Теперь, сделав в ASP обращение к методу ShowRecordSetASP:

```
objRSServer. ShowRecordSet(Response,    _
  "Provider=Microsoft.Jet.OLEDB.3.51;" & _
  "Persist Security Info=False;" & _
  "Data Source=C:\vb-db\nwind.mdb",  _
  "Select FirstName +' ' + LastName as Имя_Фамилия," & _
  "Title as Должность, " & _
  "BirthDate as ДатаРождения, HomePhone as ДомашнийТелефон" & _
  " from Employees Order by LastName, FirstName", _
  "Список сотрудников"
```

с помощью браузера можно будет посмотреть полезную информацию, хранимую на Web-сервере (рис. 329).

[57] http://www.visual.2000.ru/develop/ms-vb/tips/0011add.htm

Список сотрудников			
Имя_Фамилия	Должность	ДатаРождения	ДомашнийТелефон
Steven Buchanan	Sales Manager	04.03.55	(71) 555-4848
Laura Callahan	Inside Sales Coordinator	09.01.58	(206) 555-1189
Nancy Davolio	Sales Representative	08.12.48	(206) 555-9857
Anne Dodsworth	Sales Representative	27.01.66	(71) 555-4444
Andrew Fuller	Vice President, Sales	19.02.52	(206) 555-9482
Robert King	Sales Representative	29.05.60	(71) 555-5598
Janet Leverling	Sales Representative	30.08.63	(206) 555-3412
Margaret Peacock	Sales Representative	19.09.37	(206) 555-8122
Michael Suyama	Sales Representative	02.07.63	(71) 555-7773

Рис. 329

Совет 330. Как отлаживать ASP-скрипты

Отладить процедуры для создания ASP-страниц непросто: необходимо писать код, потом создавать DLL, затем обращаться из браузера к ASP. Увидев ошибки, найти их причину, исправить в VB программу, создать DLL... С этой проблемой мы столкнулись при разработке довольно простой процедуры ShowRecordSetASP (см. совет 329[58]), решив ее следующим образом. Мы сделали стандартный VB-проект, в котором создали процедуру ShowRecordSet такого вида:

```
Public Sub ShowRecordSet(strConnectString$, strSQL$, strHeading$)
...
  Nf=FreeFile
  Open "d:\file.htm" For Output As #Nf
  Print #Nf, "<html><head></head><body>"
  ' формирование таблицы
  Print #Nf, "<table cellpadding=3 border=0><tr>"
 ...  ' мы опустили код, который приведен в ShowRecordSet

  Print #Nf, "</body></html>"
  Close #Nf
End Sub
```

Так выглядит процедура, формирующая нужный фрагмент кода в виде готового HTM-файла, который сразу можно просматривать браузером. Затем после отладки мы убрали строчки:

```
Open "d:\file.htm" For Output As #Nf
Print #Nf, "<html><head></head><body>"
...
Print #Nf, "</body></html>"
Close #Nf
```

[58] http://www.visual.2000.ru/develop/ms-vb/tips/0011.htm

и в режиме Replace заменили "Print #1," на ".Write", а потом давили логические скобки:

```
With ASPResponse
   ...
End With
```

Существует еще один вариант разработки кода для ASP-страниц. Делаем такую процедуру, которая формирует автономный HTM-файл:

```
Public Sub MyShowRecordSet(strConnectString$, strSQL$, strHeading$)

  Nf=FreeFile
  Open "c:\file.htm" For Output As #Nf
  Print #Nf, "<html><head></head><body>"
  Call MyShowRecordSetMain(strConnectString$, strSQL$, strHeading$)
  Print #Nf, "</body></html>"
  Close #Nf
```

Подпрограмма MyShowRecordSetMain записывает нужный код в HTM-файл. Для ASP пишем еще пару подпрограмм:

```
Public Sub MyShowRecordSetASP _
  (ASPResponse As ASPTypeLibrary.Response, _
  strConnectString$, strSQL$, strHeading$)

  Nf=FreeFile
   FileName$ = "c:\TempFile.htm"
  ' формируем промежуточный текстовый файл
  Open FileName$ For Output As #Nf
  Call MyShowRecordSetMain(strConnectString$, strSQL$, strHeading$)
  Close #Nf
  ' переписываем его в ASP
  Call CopyFromFileToASP(FileName$, ASPResponse)
  Kill FileName$    ' удалить временный файл
End Sub

Public Sub CopyFromFileToASP(FileName$, _
      ASPResponse As ASPTypeLibrary.Response)
  ' переписываем содержимое текстового файла в ASP
  Dim a$
  nf= FreeFile
  Open FileName$ For Input As #Nf
  Do while not eof(Nf) ' переписываем
    Line Input #Nf, a$
    ASPResponse.Write a$
  Loop
  Close #Nf
End Sub
```

Совет 331. Как улучшить читаемость кода

Качество оформления программного кода — один из важных факторов, влияющих на эффективность разработки приложений. Прежде, когда программист имел дело с бумажными распечатками кода, считалось, что одна процедура не должна превышать 60-90 строк кода (1-1,5 страницы). Такой объем текста человек может охватить одним взглядом и, таким образом, легко воспринять логику работы данного фрагмента. С внедрением персональных компьютеров необходимость в бумажных листингах отпала, наступило время интерактивной работы с кодом с помощью экрана монитора.

В текстовом режиме экрана (когда мы работали с QuickBasic) мы придерживались правила, что объем процедуры не должен превышать 40-50 строк, то есть занимать не более двух экранных страниц, которые можно легко перелистывать «туда-сюда». Теперь, при работе в графическом режиме программист может гибко управлять размером шрифта и окна с кодом. Но чаще всего приведенные выше пределы числа строк можно рекомендовать и в этом случае.

С размером текста процедур непосредственно связан вопрос навигации между отдельными компонентами VB-проекта. Следует открыто признать, что навигация в VB оставляет желать лучшего. К сожалению, сегодня многие VB-программисты даже не представляют себе, что этот процесс можно сделать более удобным. Довольно удачным примером этого может служить окно выбора нужной процедуры, которое было реализовано в VB 3.0 (рис. 331).

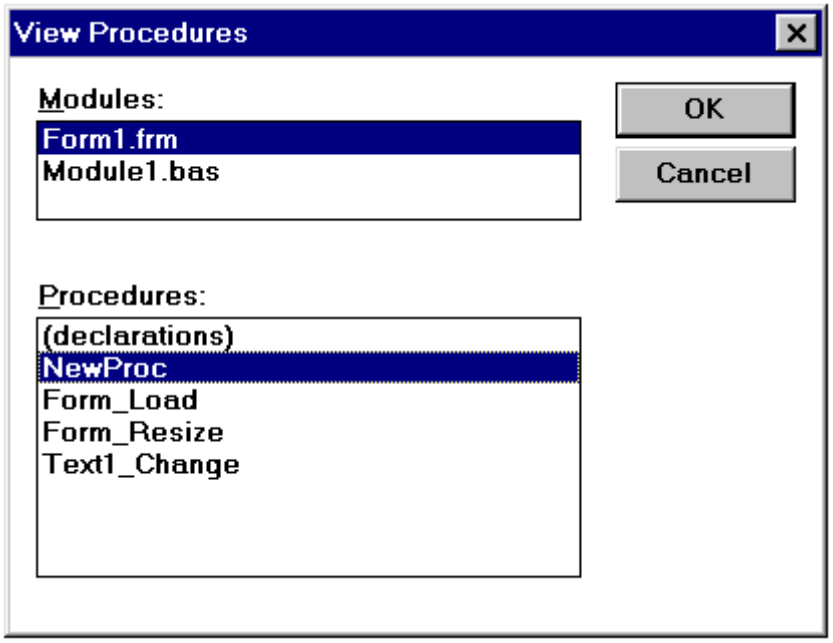


Рис. 331

Оно гораздо удобнее существующего сегодня Object Browser хотя бы потому, что для модуля формы показываются все реально сформированные процедуры.

Из сказанного выше понятно, что повышение удобства читаемости программного кода связано, в частности, с уменьшением числа строк текста. В этой связи хотелось бы напомнить о возможностях синтаксиса VB, о которых, как выясняется, многие программисты даже не подозревают.

1. Возможность написания нескольких операторов в одной строке. Для этого используется разделитель операторов в виде двоеточия. Согласитесь, что такой код:

```
2.   A = 1: C$ = "Строковая переменная": D = #09/30/2000#
```

или:

```
Select Case MyVar
   Case 0: OtherVar = 1
   Case 1: OtherVar = 45
End Select
```

который читается достаточно хорошо, но при этом занимает меньше строк.

3. Оператор If ... Then ... Else может писать в одну строку, причем в последнем случае не нужно использовать закрывающую скобку End If. Вот несколько примеров использования этой возможности:

   Вместо

```
If k = 0 Then
   A =1
End If
```

   можно написать

```
If k = 0 then A = 1
```

   Вместо:

```
If k = 0 Then
  A = 1
  B = 2
Else
  A = 2
  B = 1
End If
```

   можно написать:

```
If k = 0 Then A = 1: B = 2 Else A = 2: B = 1
```

   или:

```
If k = 0 Then A = 1: B = 2 _
    Else A = 2: B = 1
```

Совет 332. Проверка на недействительные символы

С этой задачей приходится сталкиваться довольно часто. Самый простой пример — ввод строки, которая должна содержать число. Ранее мы уже приводили простые процедуры собственной разработки, но сейчас хотим показать возможности библиотеки SHLWAPI.DLL, которая входит в состав Internet Explorer:

```
Declare Function StrSpn Lib "SHLWAPI" Alias _
   "StrSpnW" (ByVal psz As Long, ByVal pszSet As Long) As Long

Public Function IsDecimal (ByVal sString As String) As Boolean
  Const DECIMAL_NUM As String = "0123456789"
  Dim iPos

  iPos = StrSpn (StrPtr(sString), StrPtr(DECIMAL_NUM)
  ' если возвращается значение, не равное длине исходной строки,
  ' то значит найдены символы, не являющиеся цифрами
  IsDecimal = (iPos = Len(sString))
End Function
```

Совет 333. Проверка дубликатности элементов списка

При добавлении нового элемента списка полезно проверить, не содержит ли он уже такие строки. Это можно сделать с помощью следующего кода:

```
Declare Function SendMessageByString _
   Lib "User32" Alias "SendMessageA _
   (ByVal hWnd As Long,    _
   ByVal wMsg As Long, _
   ByVal wParam As Long, _
   ByVal lParam As String) As Long

   ...
   NewListItem$ = "Новый элемент списка"
   If SendMessageByString _
       (List1.hWnd, &H1A2,    -1, NewListItem$) = - 1 then
     List1. AddItem          ' добавить новый элемент
   Else
     MsgBox NewListItem$ & "- такой    элемент уже есть в списке"
   End If
```

Совет 334. Как организовать выбор каталога

Vы уже приводили пример решения этой задачи с помощью обращения к DLL-функциям (см. Совет 205[59]). Наш читатель Сергей Мичковский предложил более компактный вариант решения этой задачи:

```
Sub Main()
  Dim oShell As New Shell32.Shell
  Dim oFolder As Shell32.FolderSet
  oFolder = oShell.BrowseForFolder _
    (UserControl.hWnd, "Выберите компьютер и нажмите кнопку OK", _
    &H1000, &H12)
  MsgBox oFolder.Title
End Sub
```

Андрей Колесов, Ольга Павлова

© Андрей Колесов, Ольга Павлова, 2000
Авторский вариант. Статья была опубликована с незначительной литературной правкой в журнале "КомпьютерПресс" N 12/2000, CD-ROM.

Совет 335. Как передать параметры между процедурами

Мы уже отмечали (см. Совет 217а[60]), что передача данных между формами Visual Basic представляет некоторую проблему в связи с тем, что эти формы не могут хранить у себя глобальные переменные. К приведенным ранее рекомендациям по организации информационного взаимодействия форм предлагаем еще один вариант, который использует процедуры Property Let и Get, реализующие новые свойства формы.

[59] http://www.visual.2000.ru/develop/ms-vb/tips/9910.htm
[60] http://www.visual.2000.ru/develop/ms-vb/tips/9912.htm

Например, вы хотите, щелкнув мышью на форме Form1, загрузить форму Form2 и передать ей некоторый числовой параметр. Для этого включите в Form2 следующий код:

```
Private myVar as Variant

Public Property Get PassVar() As Variant
   PassVar = myVar
End Property

Public Property Let PassVar(ByVal vNewValue As Variant)
   If IsNumeric(vNewValue) Then myVar = vNewValue
End Property
```

Разумеется, вы можете установить более точный тип свойства, но такие изменения нужно сделать в обеих процедурах — Get и Let. Обратите также внимание, что мы включили код проверки для определения, является ли передаваемое значение числом. Теперь, чтобы записать или прочитать значение свойства, используйте следующий код в форме Form1:

```
Private Sub Command1_Click()
  With Form2
    .PassVar = 25          ' записать в Form2
    .Show                  ' показать Form2
     Debug.Print .PassVar  ' прочитать значение
  End With
End Sub
```

Совет 336. Формирование сокращенного имени каталога

Довольно обычной является ситуация, когда нужно выводить на форму (например, в виде метки) имя каталога, но при этом длина его полного названия выходит за пределы отведенного места. В этом случае можно использовать прием, когда из названия удаляют имена каталогов, начиная от корневого. То есть реализуется такой вариант вывода:

```
d:\My-sites\Visualmy\kolesov\kolesov.htm   ' полное имя
d:\...almy\kolesov\kolesov.htm   ' имя, выданное на экран
```

Для преобразования имен можно использовать следующий код:

```
Public Function PathNameShort$(Lens%, PathNameFull$)
   ' формирование сокращенного имени каталога
   'ВХОД:
   ' Lens - макс. длина названия
   ' PathNameFull$ - полное имя каталога
   'ВЫХОД:
   ' PathNameSmall$ - сокращенное имя для вывода
  If Len(PathNameFull$) <= Lens% Then   ' поместилось полностью
    PathNameShort$ = PathNameFull$
  Else ' убираем лишние символы
    PathNameShort$ = Left$(PathNameFull$, 3) + "..." + _
    Right$(PathNameFull$, Lens% - 6)
  End If
End Function
```

В данном случае пользователь сам задает максимальное число символов в имени. Но как согласуется число символов с размером той же метки? Здесь возможны два варианта ответа:

1. В принципе, можно написать код, который в цикле будет определять физическую длину выводимого текста (об этом см. совет 147[61]), сравнивать с длиной окна и удалять из имени по одному символу, пока не будет получена заданная длина текста.
2. Установив для метки шрифт с равномерной шириной символов (например, Courier New), экспериментальным путем можно легко определить, сколько символов будет помещаться в нем. В этом случае функция PathNameShort подойдет без изменений.

Совет 337. Инициализация ActiveX DLL

Как сделать, чтобы при загрузке ActiveX DLL была бы выполнена динамическая инициализация ее данных, объектов или какие-то другие операции, например анализ наличия необходимых системных ресурсов? Для этого нужно просто создать в DLL модуль кода BAS, включить в него процедуру Main и указать ее в качестве стартовой в окне свойств проекта. При загрузке библиотеки управление будет передано именно этой подпрограмме.

Совет 338. Как определить события инициализации и завершения работы ActiveX DLL

Например, нужно контролировать время работы ActiveX DLL, и вы хотите, чтобы она сама записывала куда-то время своей загрузки и выгрузки. Учитывая предыдущий совет[62], с загрузкой дело обстоит очень просто — нужно использовать подпрограмму Main. А как быть с выгрузкой? Можно реализовать и такой вариант.

Создайте тестовую ActiveX DLL с именем LbEvent.DLL, в которой находится несколько объектов (классов), в том числе clsTest. Теперь добавьте к этому проекту вспомогательный модуль класса libEvent и добавьте такой код в его событийные процедуры:

```
Private Sub Class_Initialize()
  MsgBox "Запуск DLL"
End Sub

Private Sub Class_Terminate()
      MsgBox "Выгрузка DLL"
End Sub
```

А в стартовую подпрограмму Main добавьте следующий код:

```
Private Sub Main()
   ' инициализация DLL
   MsgBox "Sub Main - DLL загружена"
   Static Lib As LibEvent
   Set Lib = New LibEvent
End Sub
```

При запуске библиотеки создается статический экземпляр класса LibEvent (и выполняется его процедура Initialize). Поскольку он статический, то не может быть закрыт до тех пор, пока библиотека не будет выгружена. В момент выгрузки выполнится процедура Terminate.

[61] http://www.visual.2000.ru/develop/ms-vb/tips/9812.htm
[62] http://www.visual.2000.ru/develop/ms-vb/tips/0012.htm

Сформируйте DLL и затем создайте тестовое приложение (не забудьте сделать в окне Reference ссылку на LbEvent.DLL):

```
Dim MyLib As Object

Private Sub Form_Click()
  Set MyLib = New Lbivent.clsTest
End Sub
```

После запуска приложения щелкните мышью по его форме: для создания объекта clsTest выполнится загрузка библиотеки, и вы увидите сначала сообщение "Sub Main - библиотека загружена" (выполнилась процедура Main), а затем "Запуск DLL" (инициализация модуля LibEvent). Закройте форму, и вы увидите сообщение "Выгрузка DLL".

Совет 339. Как определить полное имя Host-приложения

Предположим, вы создали собственный элемент управления (UserControl) и хотите, чтобы он смог узнать каталог, где находится его родительское приложение (откуда он был вызван в данный момент). В общем случае можно просто предусмотреть передачу этих сведений с помощью установки соответствующих свойств элемента управления. Но иногда бывает удобнее использовать код, который определит полный путь родительского EXE-модуля, независимо от того, захотела вызывающая программа сделать такую установку или нет.

Для этого включите в свой элемент управления следующий код:

```
 Private Declare Function GetModuleFileName Lib _
   "kernel32" Alias "GetModuleFileNameA" (ByVal _
   hModule As Long, ByVal lpFileName As String, ByVal _
   nSize As Long) As Long

Private Sub UserControl_Paint()
    ' определить каталог родительского приложения

  Dim AppPath As String
  Const MAX_PATH = 260

  UserControl.Cls
  AppPath = Space$(MAX_PATH)
  If GetModuleFileName(0, AppPath, Len(AppPath)) Then
    AppPath = Left$(AppPath, InStr(AppPath, vbNullChar) - 1)
    UserControl.Print AppPath
  Else
    UserControl.Print "Не смогли определить!"
  End If
End Sub
```

Совет 340. Как ввести табуляцию в Textbox

При работе с элементом управления Rich Textbox порой бывает необходимо использовать табуляторы при вводе данных. Если Rich Textbox является единственным элементом на форме, то нажатие клавиши Tab как раз вставляет табулятор в текст. Но если на форме находится несколько элементов управления, то нажатие клавиши Tab будет приводить к перемещению фокуса между ними. Как же быть? Очень просто — нажимайте Ctrl+Tab и табулятор будет вставляться в любом случае.

Вставлять табуляторы в обычное текстовое поле можно с помощью того же Ctrl+Tab, написав такой код обработки нажатия этих клавиш:

```
Private Sub TextBox1_KeyDown(KeyCode As Integer, Shift As Integer)
  ' вставка табулятора при нажатии Ctrl+Tab
  If (KeyCode = vbKeyTab) and (Shift = 2) Then
    TextBox1.SelText = vbTab
    KeyCode = 0
  End If
End Sub
```

Правда, в этом случае в поле Text1 введенный табулятор будет просто отображен специальным символом (жирная вертикальная полоска), так как данный элемент управления не знает, как нужно реагировать на этот код. Такую обработку символа табуляции придется писать вручную, например вставляя вместо табулятора заданное число пробелов:

```
TextBox1.SelText = Space(4)
```

Совет 341. Динамическая передача данных из списка в текстовое поле

Задача формулируется следующим образом: во время перемещения по списку вы хотите, чтобы в текстовом окне отображался текущий элемент списка. В совете 123[63] мы рассматривали аналогичный случай, когда хотели увидеть полностью текст, который не помещается по ширине списка. Там мы использовали динамическую установку свойства List1.ToolTipText, которое можно легко заменить на Text1.Text.

Однако тогда решение было связано со слежением за положением указателя мыши, а сейчас мы предлагаем использовать таймер. Рассмотрим это на примере работы с комбинированным списком. Установите на форме таймер с интервалом, например, в четверть секунды и в отключенном режиме по умолчанию. Далее напишите следующий код:

```
Private Sub Combo1_DropDown()
  ' при открытии списка таймер запускается
  Timer.Enable = True
End Sub

Private Timer1_Timer()
  ' каждые 250 мс выполняется обновление:
  Text1.Text = Combo1.List(ListIndex)
End Sub

Private Sub Combo1_Click()
  ' по щелчку список закрывается
  Timer.Enable = False   ' таймер отключается
  Text1.Text = Combo1.List(ListIndex) ' поле обновляется
End Sub
```

Совет 342. Как переписать удаленные файлы в Recycle Bin

Оператор Kill просто удаляет файлы. А как сделать, чтобы записать эти файлы в "Корзину"? Для этого используйте следующий код:

[63] http://www.visual.2000.ru/develop/ms-vb/tips/9803.htm

```
Private Declare Function SHFileOperation Lib _
   "shell32.dll" (ByRef lpFileOp As _
   SHFILEOPSTRUCT) As Long

Private Const ERROR_SUCCESS = 0&
Private Const FO_COPY = &H2
Private Const FO_DELETE = &H3
Private Const FO_MOVE = &H1
Private Const FO_RENAME = &H4
Private Const FOF_ALLOWUNDO = &H40
Private Const FOF_CONFIRMMOUSE = &H2
Private Const FOF_FILESONLY = &H80
Private Const FOF_MULTIDESTFILES = &H1
Private Const FOF_NOCONFIRMATION = &H10
Private Const FOF_NOCONFIRMMKDIR = &H200
Private Const FOF_RENAMEONCOLLISION = &H8
Private Const FOF_SILENT = &H4
Private Const FOF_SIMPLEPROGRESS = &H100
Private Const FOF_WANTMAPPINGHANDLE = &H20

Private Type SHFILEOPSTRUCT
   hwnd As Long
   wFunc As Long
   pFrom As String
   pTo As String
   fFlags As Integer
   fAnyOperationsAborted As Long
   hNameMappings As Long
   lpszProgressTitle As String ' можно использовать только
                          '    FOF_SIMPLEPROGRESS
   End Type

Public Sub Recycle(ByVal FileName As String)
   ' вызов Recycle
  Dim CFileStruct As SHFILEOPSTRUCT

  With CFileStruct
    .hwnd = Me.hwnd
    .fFlags = FOF_ALLOWUNDO
    .pFrom = FileName
    .wFunc = FO_DELETE
  End With

  If SHFileOperation(CFileStruct) <> ERROR_SUCCESS Then
    ' произошла ошибка
  End If

End Sub
```

Чтобы протестировать процедуру, создайте какой-нибудь текстовый файл и удалите его с помощью такого кода:

```
Private Sub Command1_Click()
  Recycle "c:\test.txt"
End Sub
```

Совет 343. Поддержка контекстного меню для Rich Textbox

VB предлагает достаточно простые способы создания контекстных меню, но некоторые элементы управления, в частности Rich Textbox, не включают их поддержку. Чтобы исправить это упущение, можно использовать событие MouseDown.

Для тестирования такой ситуации добавьте к форме элемент управления Rich Textbox, затем с помощью Menu Editor создайте меню из пары строк команд. Введите такой код в событийную процедуру:

```
Private Sub RichTextBox1_MouseDown(Button As Integer, _
  Shift As Integer, X As Single, Y As Single)
  If Button = vbRightButton Then Me.PopupMenu Menu
End Sub
```

Запустите проект, щелкните правой кнопкой мыши на элементе Rich Textbox и убедитесь, что появилось меню.

Совет 344. Дайте возможность пользователям изменять размеры элементов управления

Вы можете предоставить пользователям своего приложения возможность изменять размеры элемента управления с помощью мыши — аналогично тому, как вы это делаете в режиме разработки. Для этого необходимо только вызвать две простые функции Windows API — ReleaseCapture и SendMessage. Если задать диапазоны, в которых может двигаться мышь (например, X > 0 и X < 100), событие MouseDown активизирует эти функции и меняет размеры элемента управления при перемещении мыши.

Предположим, что вы поместили на форму картинку (PictureBox). Теперь напишите для нее следующий код:

```
Private Declare Function ReleaseCapture Lib "user32" () As Long
Private Declare Function SendMessage Lib "user32" Alias _
   "SendMessageA" (ByVal hWnd As Long, ByVal wMsg As Long, _
   ByVal wParam As Long, lParam As Any) As Long
Private Const WM_NCLBUTTONDOWN = &HA1
' В API Viewer есть и другие константы
' Заданные здесь константы используются
' только для изменения правой и левой границ
Private Const HTLEFT = 10
Private Const HTRIGHT = 11

Private Sub Picture1_MouseDown(Button As Integer, _
  Shift As Integer, X As Single, Y As Single)
  Dim nParam As Long
  With Picture1
    ' Здесь вы можете задать любые координаты
    If (X > 0 And X < 100) Then
       nParam = HTLEFT
      ' и здесь тоже
    ElseIf (X > .Width - 100 And X < .Width) Then
      nParam = HTRIGHT
    End If
    If nParam Then
      Call ReleaseCapture
      Call SendMessage(.hWnd, WM_NCLBUTTONDOWN, nParam, 0)
    End If
  End With
End Sub

Private Sub Picture1_MouseMove(Button As Integer, _
     Shift As Integer, X As Single, Y As Single)
  Dim NewPointer As MousePointerConstants
  ' Здесь вы можете задать любые координаты
```

```
  If (X > 0 And X < 100) Then
    NewPointer = vbSizeWE
    ' и здесь тоже
  ElseIf (X > Picture1.Width - 100 And X < Picture1.Width) Then
    NewPointer = vbSizeWE
  Else
    NewPointer = vbDefault
  End If
  If NewPointer <> Picture1.MousePointer Then
    Picture1.MousePointer = NewPointer
  End If
End Sub
```

Совет 345. В циклах лучше используйте GetInputState

Некоторые разработчики предлагают использовать в циклах функцию DoEvents, чтобы приложение могло реагировать на какие-либо события. Однако это не всегда целесообразно. Так, если цикл короткий, функция DoEvents и не нужна, а если длинный, то использование этой функции может нанести ущерб быстродействию приложения. А что же делать, если необходимо, чтобы пользователь имел возможность щелкнуть мышью на кнопке Cancel (Отмена) или выполнить какое-либо другое действие в процессе выполнения цикла?

Решение этой проблемы возможно путем использования функции Windows API GetInputState, которая возвращает 1, если пользователь щелкает на кнопке мышью или нажимает клавишу на клавиатуре. Применение этой функции требует намного меньше ресурсов, чем при работе с DoEvents, поэтому цикл выполняется быстрее.

Однако если щелчок мыши или нажатие клавиши инициирует выполнение некоторого события, то следует вызывать функцию DoEvents. Другими словами, вызывайте DoEvents только в том случае, если вам действительно необходимо обработать это событие. При этом вы можете уменьшить использование ресурсов, осуществляя проверку каждые x итераций (конкретное число зависит от того, сколько времени занимает выполнение цикла):

```
Private Declare Function GetInputState Lib "user32" () As Long
Private m_UserCancel As Boolean

Private Sub cmdCancel_Click()
   m_UserCancel = True
End Sub

Private Sub cmdGo_Click()
  Dim lCounter As Long
  m_UserCancel = False
  Me.MousePointer = vbHourglass
  For lCounter = 0 To 10000000
    ' любой длинный цикл,
    ' который необходимо прервать
    If lCounter Mod 100 Then
      If GetInputState <>    0 Then
        ' событие,    инициируемое щелчком мыши
        ' или    нажатием клавиши, находится в
        'очереди    сообщений, поэтому вызываем
        ' функцию    DoEvents,чтобы начать обработку
        ' этого    события
        DoEvents
        If m_UserCancel    Then Exit For
```

```
      End If
    End If
  Next lCounter
  Me.MousePointer = vbDefault
End Sub
```

Совет 346. Работа с Интернетом в VB-приложениях

Создавая любое современное клиентское Web-приложение, вы, вероятно, захотите, чтобы у пользователи имели при работе с ним имели выход в Интернет. Для того чтобы реализовать такую возможность, разработчики обычно используют элемент управления WebBrowser (для его установки выберите команду Project|Components, а затем — элемент Microsoft Internet Controls). Однако у пользователей при этом обязательно должен быть установлен Internet Explorer, иначе приложение не запустится. Чтобы решить подобную проблему, удалите элемент управления WebBrowser из приложения, а затем загрузите его динамически, выяснив, что у пользователя установлен Internet Explorer. Для этого напишите следующий код:

```
Private ie As VBControlExtender

Private Sub Form_Load()
  On Error GoTo IEMissing
  Set ie = Form1.Controls.Add("Shell.Explorer", "wcIE")
  ie.Visible = True
IEMissing:
  ' обработка ошибки
End Sub

Private Sub Form_Resize()
  If Not (ie Is Nothing) Then
    ie.Move 0, 0, Me.ScaleWidth, Me.ScaleHeight
  End If
End Sub
```

Вы можете совершать самые разнообразные операции с этим объектом, например менять его видимость, но при этом вам не будут доступны уникальные свойства и методы Internet Explorer. Так, если вы введете obj.Navigate sMyURL, VB даст сообщение о том, что данный объект не поддерживает указанное свойство или метод. Секрет заключается в том, что свойство Object объектной переменной следует использовать только подобным образом:

```
Private Sub Form_Activate()
  If Not (ie Is Nothing) Then
    ie.object.Navigate "http://www.visual.2000.ru"
  End If
End Sub
```

Совет 347. Как определить текущие размеры экрана

Последние модели видеодрайверов позволяют менять разрешение экрана без перезагрузки компьютера. К сожалению, объект Screen не всегда правильно возвращает размеры экрана — он запоминает только те значения, которые были установлены на момент запуска приложения. Представляется, что такое поведение зависит от драйвера, однако подобная ситуация может быть вызвана и работой операционной системы (этого никогда не случится, если на вашей машине установлена Windows 98, но не Windows NT).

Поэтому если необходимо определить размерность экрана в любой другой момент работы приложения, а не только при выполнении события Form_Load, пользуйтесь функциями API, как показано ниже:

```
Private Type RECT
  Left As Long
  Top As Long
  Right As Long
  Bottom As Long
End Type

Private Declare Function GetDesktopWindow Lib _
   "user32" () As Long
Private Declare Function GetWindowRect Lib "user32" _
   (ByVal hWnd As Long, lpRect As RECT) As Long

Public Function ScreenWidth() As Single
  Dim R As RECT
  GetWindowRect GetDesktopWindow(), R
  ScreenWidth = R.Right * Screen.TwipsPerPixelX
End Function

Public Function ScreenHeight() As Single
  Dim R As RECT
  GetWindowRect GetDesktopWindow(), R
  ScreenHeight = R.Bottom * Screen.TwipsPerPixelY
End Function
```

Совет 348. Как установить связь с Microsoft Excel с помощью OLE DB

В документации Microsoft говорится, что можно установить связь с Excel 97 или Excel 2000 с помощью провайдера Microsoft.Jet.OLEDB 4.0. Однако если при этом вы используете элемент управления Microsoft ADO Data Control (Adodc), то могут возникнуть проблемы.

Продемонстрируем это на следующем примере. Поместите на форму элемент управления Adodc, во вкладке General окна свойств установите переключатель Use Connection String и щелкните кнопку Build. Затем выберите необходимую базу данных, установив Excel-файл в качестве файла базы данных. Теперь, если вы щелкните кнопку Test Connection, чтобы проверить, установилась ли связь, VB выдаст сообщение об ошибке, в котором говорится о невозможности установления связи с базой данных из-за того, что файл имеет неизвестный формат.

Однако у вас есть выход. Подтвердите получение сообщения об ошибке и вернитесь во вкладку General окна свойств. В поле ввода Use Connection String в конце строки, используемой для связи с Excel, допишите следующий код:

```
Extended Properties = Excel 8.0;
```

Тогда полная строка будет выглядеть так:

```
Provider = Microsoft.Jet.OLEDB/4.0; Data Source = FileName; _
   Extended Properties = Excel 8.0;
```

Теперь, если вы щелкнете кнопку Build, а затем — Test Connection, то ошибки не будет, и связь с базой данных Excel будет успешно установлена.

Совет 349. Простой ввод данных в элементе управления MSFlexGrid

Существует возможность осуществлять ввод данных в элемент управления MSFlexGrid, не используя другие дополнительные компоненты. Это можно реализовать с помощью событий KeyPress и KeyUp, как показано ниже. Для начала поместите на форму элемент управления MSFlexGrid, присвойте ему имя FlxGrdDemo, а затем введите следующий код:

```
Private Sub FlxGrdDemo_KeyPress(KeyAscii As Integer)

  Select Case KeyAscii
    Case vbKeyReturn
      ' Когда пользователь нажимает
      ' клавишу Enter, этот код
      ' осуществляет переход к
      ' следующей ячейке или ряду
      With FlxGrdDemo
        If .Col + 1 <= .Cols - 1 Then
          .Col = .Col + 1
        ElseIf .Row + 1 <= .Rows - 1 Then
          .Row = .Row + 1: .Col = 0
        Else
          .Row = 1: .Col = 0
        End If
      End With
    Case vbKeyBack
      ' Удаляет предыдущий символ при
      ' нажатии клавиши Backspace
      With FlxGrdDemo
        If Trim(.Text) <> " " Then _
         .Text = Mid(.Text, 1, Len(.Text) - 1)
      End With
    Case Is < 32
      ' Не разрешает вводить непечатные символы
    Case Else ' Разрешает печатать все
      With FlxGrdDemo
        .Text = .Text & Chr(KeyAscii)
      End With
  End Select
End Sub

Private Sub FlxGrdDemo_KeyUp(KeyCode As Integer, _
    Shift As Integer)
  Select Case KeyCode
    Case vbKeyC And Shift = 2 ' Ctrl + C
      ' Копирует символы
      Clipboard.Clear
      Clipboard.SetText FlxGrdDemo.Text
      KeyCode = 0
    Case vbKeyV And Shift = 2 ' Ctrl + V
      ' Вставляет символы
      FlxGrdDemo.Text = Clipboard.GetText
      KeyCode = 0
    Case vbKeyX And Shift = 2 ' Ctrl + X
      ' Вырезает символы
      Clipboard.Clear
      Clipboard.SetText FlxGrdDemo.Text
      FlxGrdDemo.Text = " "
      KeyCode = 0
    Case vbKeyDelete
      ' Удаляет символы
      FlxGrdDemo.Text = " "
  End Select
```

```
End Sub
```

Вы можете также установить свойство FillStyle как FlexFillRepeat, что позволяет вносить изменения во все выделенные ячейки.

Совет 350. Десять советов "как не нужно поступать при работе с ASP"

Следуя им, можно создавать более эффективные и надежные ASP-приложения.

1. Не следует часто переключаться от HTML-текста к коду скрипта и обратно. При создании HTML-текста с использованием большого числа изменяемых фрагментов применяйте последовательность методов Response.Write.
2. Не нужно на одной ASP-странице использовать коды JScript и VBScript. Такое смешение разных языков ограничивает число откомпилированных страниц, которые IIS может хранить в кэш-памяти.
3. Не забывайте отключить режим отладки в отлаженных Web-приложениях. При включенном режиме IIS работает как однопотоковое приложение, снижая общую производительность системы.
4. Не используйте большие или вложенные Include-файлы. В любом случае старайтесь применять Include-файлы только для загрузки процедур, которые действительно нужны для данной ASP-страницы.
5. Не используйте повторно один и тот же объект ADO Command для двух разных запросов на одной странице. Например, система может просто "рухнуть", если вы будет применять такой объект для работы с двумя разными параметризованными хранимыми процедурами.
6. Не используйте OLE DB Provider for ODBC Drivers, если у вас есть возможность доступа к базе данных через ее "родной" OLE DB-провайдер. Последний всегда работает надежнее и быстрее.
7. Не следует применять метод Response.Redirect для перехода к другой ASP-странице, находящейся на этом же сайте. Если вы работаете с IIS 5.0, то лучше воспользоваться новыми методами Server.Transfer и Server.Execute.
8. Не создавайте наборы записей (recordsets) ADO с большей, чем это нужно, функциональностью. Чем больше функциональности — тем меньше эффективности (по объему кода и быстродействию). Например, если вы не собираетесь модифицировать данные, задайте режим "только для чтения", если будете просматривать их только в одном направлении — "только вперед" и т.д.
9. Не храните большие массивы в виде переменных Application и Session. IIS копирует массив полностью, когда вы обращаетесь только к одному его элементу из скрипта. Понятно, что время копирования массива пропорционально его размеру.
10. Не используйте серверные скрипты (server-side scripts) для выполнения операций, которые можно проделать с помощью скриптов на клиентской машине. Например, проверку значений, вводимых в поля формы, лучше сделать непосредственно в браузере.

Андрей Колесов, Ольга Павлова

© Андрей Колесов, Ольга Павлова, 2001
Авторский вариант. Статья была опубликована с незначительной литературной правкой в журнале "КомпьютерПресс" N 03/2001, CD-ROM.

Совет 351. Пишите замечания, присылайте советы, задавайте вопросы

Не стоит даже повторять, как важна для авторов обратная связь с читателями. И мы бы очень хотели, чтобы программисты не только указывали нам на наши оплошности, но и делились собственными находками и "трюками".

Промашки у нас, конечно, бывают. Например, Евгений Иванов справедливо заметил, что в Совете 307[64] мы, рассказывая об удалении каталога с помощью Windows API, совсем забыли сказать, что в большинстве случаев с этой задачей отлично справляется давно знакомая встроенная Basic-функция rmDir.

Приятно отметить, что журнал "КомпьютерПресс" и наши "Советы" читают не только в нашей стране, но и в дальнем зарубежье (кстати, хотя русскоязычных программистов в Европе и США гораздо меньше, чем в России, их активность на электронных форумах и в переписке заметно выше). Михаил Эскин, например, живет в Мюнхене и, как оказалось, является давним читателем наших публикаций. При этом он отмечает: "Ваши статьи стали постоянным моим спутником, несмотря на появление специальной литературы на прилавках". Спасибо!

В Германию российские журналы приходят с задержкой, поэтому Михаил только в ноябре прислал некоторые замечания по поводу статьи "Календарь наших дел"[65], опубликованной в КомпьютерПресс N 5/2000. (Кстати, Михаил — автор серии статей о создании элементов управления в среде VB, опубликованных на сервере www.vbrussian.com[66].) Об этих замечаниях мы поговорим в последующих Советах.

Совет 352. Как решить проблему с VisData

Мы уже несколько раз отмечали, что у утилиты VisData (создание баз данных) существуют проблемы с вводом и просмотром русских текстов. Михаил Эскин отмечает, что это, скорее, проблема конфигурации конкретного компьютера, которая решается следующим образом: в разделе [FontSubstitutes] файла WIN.INI нужно добавить сверху строку Tahoma,0=Tahoma,204 и затем перезагрузить компьютер.

Действительно, после этого VisData стала нормально работать с русским текстом. Но мы все равно считаем, что в данном случае имеет место дефект VisData, который возможно устранить с помощью подобного "трюка", поскольку такой способ настройки утилиты для нормальной работы с русским языком нигде не описывается.

[64] http://www.visual.2000.ru/develop/ms-vb/tips/0008-2.htm
[65] http://www.visual.2000.ru/develop/ms-vb/cp0005/app_vb6.htm
[66] http://www.vbrussian.com/

Совет 353. Внимание при работе с булевыми типами данных

Михаил отметил также, что приведенный в статье "Календарь наших дел"[67] код для чтения/записи свойства DeleteMe (для пользовательского элемента управления Memos) при его привязке к значению флажка chkDeleteMe

```
Public Property Get DeleteMe() As Boolean
  If chkDeleteMe.Value = 0 Then
    DeleteMe = False
  Else
    DeleteMe = True
  End If
End Property

Public Property Let DeleteMe(ByVal newDelete As Boolean)
  If newDelete Then
    chkDeleteMe.Value = 1
  Else
    chkDeleteMe.Value = 0
  End If
End Property
```

Можно упростить, записав содержимое каждой из этих процедур в одну строку:

```
DeleteMe = -1 * chkDeleteMe.Value

chkDeleteMe.Value = -1 * newDelete
```

Несмотря на то что речь здесь идет вроде бы об очень частной проблеме, остановимся на ней подробнее.

Мы сами ранее говорили, что для улучшения читаемости программы желательно сокращать количество строк кода (Совет 331[68]). Однако это не должно провоцировать потерю управляемости программой и снижение ее эффективности (обратите внимание, что в Совете 331 речь шла о разных формах записи одних и тех же конструкций).

Нам кажется, что вариант, предложенный Михаилом Эскиным, приведет именно к таким негативным последствиям. По этому поводу отметим следующие моменты:

1. Мы считаем принципиально неверным неявное преобразование данных, в данном случае из Integer в Boolean и наоборот. К сожалению, VB позволяет делать это, хотя вполне вероятно, что в VB.NET (7.0) такие вещи будут запрещены.
2. Конечно, программисту полезно знать, в каком конкретном двоичном виде хранятся те или иные типы данных, но пользоваться такими знаниями нужно лишь в случае крайней необходимости.
3. Серьезной проблемой VB является отсутствие беззнаковых целых типов данных. При этом путаница часто возникает именно из-за того, что переменные типа Integer и Long на самом деле выступают в роли то чисел со знаками (в арифметических операциях и при использовании десятичных литералов), то беззнаковыми (в логических операциях и в шестадцатеричных и восьмеричных литералах).
4. Вообще говоря, практически в любой программе можно легко обойтись без использования типа Boolean, так как она является всего лишь частным случаем Integer (те же два байта для хранения информации). Более того, можно даже добиться экономии

[67] http://www.visual.2000.ru/develop/ms-vb/cp0005/app_vb6.htm
[68] http://www.visual.2000.ru/develop/ms-vb/tips/0011.htm

памяти — если использовать переменную типа Byte со значениями 0 или 1. Это легко можно сделать в "Календаре", где основная коррекция заключалась бы в замене в SQL-запроса:

```
вместо "Where DeleteMe = True"

написать — "Where DeleteMe = 1"
```

При этом автоматически решается проблема обмена данными между переменной и значением флажка, поскольку они оказываются тождественно равными (можно даже использовать значение флажка 2 — "может быть").

8. Но главное достоинство конструкции IF...Then...Else...EndIf — очевидность логики ее работы. Попробуйте мгновенно ответить, каково будет значение DeleteMe при chkValue = 0 в этом выражении:

```
DeleteMe = -1 * chkValue
```

10. В одну более короткую строку (но все же с увеличением машинных команд) можно было бы предложить вариант без неявного преобразования данных:

```
DeleteMe = (chkDeleteMe.Value = 1)

chkDeleteMe.Value = IIf (1, 0, DeleteMe)
```

Но с точки зрения "очевидности результата" такой код также не безупречен.

14. Несмотря на большое число строк в нашей конструкции IF...Then...Else...EndIf, очевидно, что этот код является самым компактным и быстрым (он занимает всего несколько машинных коротких команд). Более компактно его можно записать в таком виде:

```
If chkDeleteMe.Value = 0 Then DeleteMe = False _
   Else DeleteMe = True
```

Мы советуем использовать именно такой вариант.

Совет 354. Как обеспечить совместимость между VBA- и VB-проектами

Мы уже несколько раз отмечали, что, несмотря на всю схожесть VB и Office/VBA, у этих систем есть ряд серьезных различий, которые препятствуют прямому перенесению кода из одного вида проекта в другой и наоборот. Поэтому при написании кода, который предполагается для использования в разных системах, нужно специально тестировать возможность их использования в обоих вариантах. К сожалению, только изучая документацию, проверить это трудно.

Однако в общем случае следует иметь в виду, что VBA все же располагает более ограниченным набором функций по сравнению с VB (речь идет о встроенных возможностях самого языка, без учета объектов приложения, в котором используется VBA). Поэтому при прочих равных условиях для создания общих программных модулей (совместимых на уровне исходного кода) предпочтительнее среда VBA.

Это, в частности, касается и создания модулей формы. Мы уже писали (Совет 301), что Office/VBA использует для создания форм ActiveX-конструктор Microsoft Form 2.0, который доступен также в VB. То есть VB может создавать два типа: собственные VB-формы (Ruby Forms) и UserForms (VBA Forms). Однако проблема заключается в том, что, даже используя одинаковый конструктор, VB и VBA сохраняют модули формы в разных форматах. При этом VB может читать оба формата, а VBA — только свой собственный.

Соответственно, если вы намерены создавать модули формы двойного применения, это следует делать не просто с помощью MS Forms 2.0 , а обязательно с этой целью использовать Office/VBA.

И еще один совет, который из этого следует: для лучшей совместимости кода нужно в максимальной степени выносить код из процедур модулей формы в процедуры модуля кода или класса (эти компоненты пока — трудно сказать, что Microsoft придумает дальше — загружаются в обе среды разработки).

Перенос модулей формы в случае их несовместимости можно сделать следующим образом (например, из VB в VBA). Создайте в VBA визуальную форму со всеми компонентами. Задайте имена компонентов такие же, как в VB. Далее скопируйте содержимое кода из VB в VBA через буфер обмена.

Однако этот способ будет работать только при использовании Forms 2.0. При переносе VB-форм придется вручную корректировать имена некоторых событий и свойств. Например, в VBA события формы Initialize и Terminate соответствуют событиям Load и Unload в VB. В общем, с Microsoft не соскучишься.

Совет 355. Используйте свойство Button для элемента управления DataGrid

Элемент управления DataGrid позволяет установить для ячеек одной или нескольких колонок таблицы свойство Button, которое обеспечивает их работу в режиме "кнопок". Например, установите

```
DataGrid1.Columns.Item(1).Button = True
```

В этом случае после щелчка мыши по ячейкам первой колонки будет выполняться событие ButtonClick. Программист может написать в этой процедуре любой специальный код, например вывести диалоговое окно с информацией (список, таблица и пр.), которая связана с данной ячейкой.

Совет 356. Как сделать Help для своего приложения

Программ создания HELP-файлов довольно много, ряд из них — свободно распространяемые или условно-бесплатные. Для VB и VBA, возможно, лучшим способом является использование утилиты Microsoft HTML Help Workshop, которая поставляется в составе ряда программных продуктов, в том числе MS Office 2000 Developer Edition. При желании ее можно скачать из Интернета[69]. Описания работы этой утилиты имеются в целом ряде книг по VB.

Совет 357. Преобразование текстового файла в набор данных ADO

Информацию из текстового файла, записанную в виде полей, разделенных запятыми, можно достаточно просто представить в виде набора данных ADO. Это достигается следующим образом:

```
connCSV.Open "Provider=Microsoft.Jet.OLEDB.4.0;" & _
  "Data Source=" & FileName$ & _
  ";Extended Properties='text;FMT=Delimited'"
```

[69] http://www.microsoft.com/workshop/author/htmlhelp/

В данном случае строка с параметрами соединения (Connection String) содержит раздел Extended Properties, который указывает, что используется текстовый файл с полями. Однако следует иметь в виду, что приведенный вариант обращения подразумевает наличие в первой строке текстового файла заголовков полей. Если же заголовков нет, следует указать в явном виде аргумент HDR:

```
connCSV.Open "Provider=Microsoft.Jet.OLEDB.4.0;" & _
  "Data Source=" & FileName$ & _
  ";Extended Properties='text;HDR=NO;FMT=Delimited'"
```

Совет 358. Как расширить массив элементов управления во время выполнения

Порой требуется добавить элемент управления, например командную кнопку, к массиву подобных компонентов во время выполнения программы. Для этого можно использовать оператор Load:

```
Load object(index)
```

где object — имя массива, а index — номер нового элемента управления, который вы хотите добавить. Но при этом в исходном состоянии в массиве уже должен иметься хотя бы один элемент управления (нумерация индекса начинается с нуля). В более общем случае создание новой кнопки может выглядеть так, как описано ниже.

Создайте новый проект и добавьте к форме командную кнопку. Затем в окне Properties введите 0 для свойства Index — VB сразу преобразует одиночную кнопку в массив. Далее введите такой код для формы:

```
Private Sub cmdBtn_Click(Index As Integer)
  ' создание новой кнопки для массива элементов управления
  Dim btn As CommandButton
  Dim iIndex As Integer
  iIndex = cmdBtn.Count  'текущее числе элементов массива
  If iIndex <= 32767 Then '  можно добавлять
    Load cmdBtn(iIndex)
    Set btn = cmdBtn(iIndex)
    With btn  ' установка свойств
      .Top = cmdBtn(iIndex - 1).Top + 620
      .Caption = "Command" & iIndex + 1
      .Visible = True
    End With
    Set btn = Nothing
  End If
End Sub
```

Совет 359. Работа с реестром и INI-файлами с помощью System.PrivateProfileString

Работа с реестром Windows может выполняться не только с помощью функций Windows API или объекта Registry (см. Совет 273[70]), но и с применением свойства PrivateProfileString объекта System, который входит в состав библиотеки Microsoft Word 8.0/9.0 Object Library. Она автоматически подключается при работе с MS Word 97/2000 и может использоваться в любых инструментах, которые поддерживают работу с ActiveX-

[70] http://www.visual.2000.ru/develop/ms-vb/tips/0006-2.htm

объектами. В частности, к среде VB или MS Office/VBA она подключается с помощью команды Project|Reference или Tools|Reference соответственно.

Вот как будет выглядеть чтение полного имени каталога, где находится Internet Explorer:

```
RegFile$ = "" ' пустое имя означает Системный Реестр SectionName$ =
"HKEY_CURRENT_USER\Software\Microsoft\" _
    & "Windows\CurrentVersion\App Paths\IEXPLORER.EXE") ' имя раздела
KeyName$ = "Path"  ' имя ключа
IEPath$ = System.PrivateProfileString(RegFile$, SectionName$, KeyName$)
If IEPath$ <> "" Then   ' есть имя каталога
  MsgBox "Имя каталога с IE = " & IEPath$
End If
```

Соответственно запись нового значения параметра в реестр выполняется следующим образом:

```
System.PrivateProfileString(RegFile$, SectionName$, KeyName$)= IEPath$
```

Дополнительным преимуществом данного свойства является возможность работы не только с реестром, но и с любыми текстовыми файлами формата типа Win.INI, то есть возможность использовать для хранения параметров приложения персональные файлы-профайлы. Для этого достаточно просто задать в качестве первого параметра свойства имя соответствующего файла.

Например, при закрытии текущего документа Word можно автоматически фиксировать имя последнего использовавшегося документа:

```
System.PrivateProfileString("C:\MyWordSetting.ini", "MacroSettings", _
    "LastFile") = ActiveDocument.FullName
```

А при загрузке Word можно автоматически открыть данный файл:

```
LastFile$ = System.PrivateProfileString("C:\Settings.Txt", _
    "MacroSettings", "LastFile")
If LastFile$ <> "" Then Documents.Open FileName:=LastFile$
```

Необходимо обратить внимание на следующие особенности применения свойства PrivateProfileString:

- При чтении пользователь не может точно определить причину получения пустого значения ключа — это может быть как наличие пустой записи, так и отсутствие ключа, раздела или даже самого файла.
- При записи замена значения ключа выполняется только в случае, если ключ найден. В противном случае формируется новая запись с заданным ключом и его значением. При этом при отсутствии раздела создается новый раздел, при отсутствии файла — создается файл.

Например, если файл D:\MyFile.INI не существует, то после выполнения кода:

```
System.PrivateProfileString("d:\ MyFile.INI", "test1", "key1") = "andy"
```

Будет сформирован файл следующего содержания:

```
[test1]
```

```
key1=andy
```

Андрей Колесов, Ольга Павлова

© Андрей Колесов, Ольга Павлова, 2001
Авторский вариант. Статья была опубликована с незначительной литературной правкой в журнале "КомпьютерПресс" N 03/2001, CD-ROM.

Совет 360. Вместо DoEvents отслеживайте реальные события

Оператор DoEvents позволяет выполнять параллельные процессы, поэтому достаточно часто используется для синхронизации двух различных вычислительных процессов.

Типичным случаем является такой пример. Имеются две формы: главная (frmMain) выполняет некоторые вычисления и выводит результаты, а вторая (frmEntry) вводит исходные данные для этих вычислений. При этом логика взаимодействия данных форм такова: главная форма запускает frmEntry и ожидает, когда там будут введены нужные данные, например, в виде нажатия пользователем кнопки Submit (Подтверждение).

Один из вариантов решения этой задачи может выглядеть следующим образом:

```
Private Sub Command1_Click()
  ' процедура в форме frmMain
  Dim Myform As frmEntry
  ' создание второй формы для ввода данных
  Set Myform = New frmEntry
  With Myform
    .Show
    .Ready = False  ' начальная установка глобальной переменной
    Do  ' ожидание
      DoEvents  ' передача управления операционной системе
                ' для обработки других событий
    Loop Until .Ready
    'Выполнение каких-то вычислений на основе введенных данных
    txtResults.Text = .txtNum1 * .txtNum2
  End With
  Unload Myform
  Set Myform = Nothing  ' освободить объект
End Sub
'
'============
' код формы frmEntry1
Public Ready As Boolean
Private Sub cmdSubmit_Click()
  Ready = True  ' подтверждение ввода
End Sub
```

Данная конструкция базируется на отслеживании состояния глобальной переменной Ready в форме frmEntry1. Однако недостатком этой конструкции является как раз использование оператора DoEvents, который требует достаточно много времени, то есть "съедает" значительную часть ресурсов.

Гораздо лучше выглядит такой вариант решения, когда создается пользовательское событие, управляемое из формы frmEntry. Для этого в данной форме нужно написать следующий код:

```
' описание события в секции Declaration
Public Event NumbersSubmitted(Num1 As Integer, Num2 As Integer)

' выполнение операций
Private Sub cmdSubmit2_Click()
  Dim Num1%, Num2%
  Num1 = CInt(txtNum1.Text)
  Num2 = CInt(txtNum2.Text)
  Unload Me
  ' запуск внешнего события с передачей параметров
  RaiseEvent NumbersSubmitted(Num1, Num2)
End Sub
Соответственно в главной форме нужно описать данное событие и сформировать
процедуру его обработки:
Private WithEvents frmNumEntry As frmEntry

Private Sub Command2_Click()
  ' запуск второй формы
  Set frmNumEntry = New frmEntry
  frmNumEntry.Show
End Sub

Private Sub frmNumEntry_NumbersSubmitted(Num1%, Num2%)
  ' обработка события, инициализированного из формы frmEntry
  txtResults.Text = Num1 * Num2
  Set frmNumEntry = Nothing
End Sub
```

Совет 361. Избегайте неявного преобразования типов данных

Мы неоднократно подчеркивали в своих публикациях, что причиной достаточно частых ошибок является использование неявного преобразования типов данных (см. например, Совет 353[71]). Подобная возможность (без применения специальных функций преобразования) является большим недостатком VB, а ее использование разработчиками говорит в первую очередь об их небольшом опыте...

Здесь существует достаточно много подводных камней, главный из которых — неопределенность такого рода преобразования, то есть тот факт, что программа будет вести себя совсем не так, как видится ее автору. Следует также учитывать особенности национальных форматов представления данных (для вещественных чисел и дат). Приведем еще один пример на данную тему, реализованный в Windows с русскими региональными установками.

Выполните такой код:

```
Dim strSource As String, strR1 As String, strR2 As String
Dim sngResult As Single
strSource = "2.34"
' преобразование строки в вещественное число
sngResult = Val(strSource)
Print sngResult   ' будет напечатано 2,34
'
```

[71] http://www.visual.2000.ru/develop/ms-vb/tips/0103-1.htm

```
strR1 = Str(sngResult)
strR2 = sngResult
Print strR1, strR2    ' будет напечатано 2.34  2,34
```

Если вы попробуете выполнить код:

```
strSource = "2.34"
sngResult = strSource
```

то получите на втором операторе сообщение об ошибке — неверный тип данных. Далее выполните еще один код:

```
strSource = "2,34"
sngResult = Val(strSource)
Print sngResult   ' будет напечатано 2 ! Ошибка

sngResult = Val(strSource)
strR1 = Str(sngResult)
strR2 = sngResult
Print strR1, strR2    ' будет напечатано 2.34  2,34
```

Из проведенных экспериментов можно сделать следующие выводы:

1. Функции явного и неявного преобразования данных работают по-разному! Val и Str выполняют операции преобразования по правилам американских региональных установок, независимо от установок пользователя на его компьютере. Неявное преобразование выполняется с учетом установленных на компьютере параметров.
2. Казалось бы, оба варианта имеют свои недостатки. Более того, на "русской" системе лучше использовать неявное преобразование. Но здесь стоит обратить внимание на следующий момент: функции Val и Str будут одинаково работать на любом ПК, тогда как операции неявного преобразования могут выдавать разные результаты в зависимости от региональных установок.

Совет 362. Как узнать список папок Outlook

Допустим, вы хотите собрать список папок в некий список. Казалось бы, узнать имена папок можно следующим образом:

```
Dim AllFolders As Folders
Set AllFolders = Application.GetNamespace("MAPI").Folders
MsgBox AllFolders.Count
For i = 1 To AllFolders.Count
  MsgBox AllFolder.Item(i).Name
Next
```

Однако выясняется, что у вас имеется всего две папки с именами Personal Folders. Здесь полезно вспомнить, что папки OutLook имеют иерархическую структуру, такую же, как знакомая файловая система. В частности, она может иметь вид, представленный на рис.362-1. Приведенная выше конструкция выдала нам имена стандартных папок самого верхнего уровня.

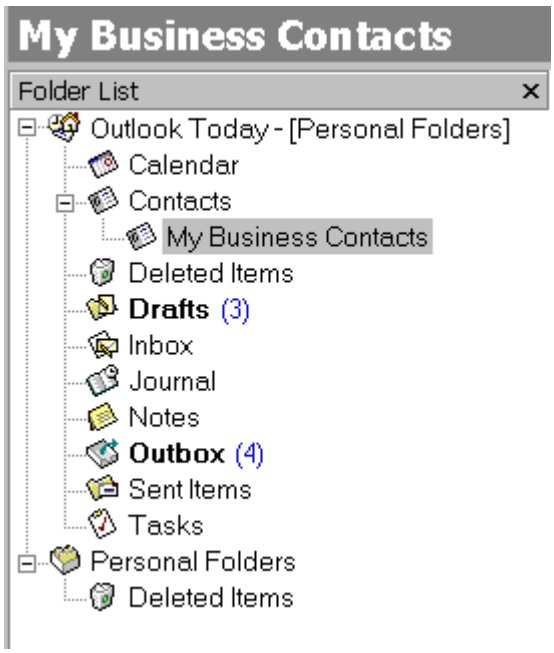


Рис. 362-1

Чтобы получить информацию о папках второго уровня, нужно написать более сложный код:

```
Dim allFolders As Folders
Dim i%, j%
Set allFolders = Application.GetNamespace("MAPI").Folders
MsgBox "Число папок верхнего уровня = " & allFolders.Count
' обзор папок верхнего уровня
For i = 1 To allFolders.Count
  MsgBox "Имя папки = " & _
    allFolders.Item(i).Name & vbCrLf & _
     "  число вложенных папок = " & _
      allFolders.Item(i).Folders.Count
  ' обзор папок второго уровня
  For j = 1 To allFolders.Item(i).Folders.Count
     MsgBox "Имена вложенной папки = " & _
       allFolders.Item(i).Folders.Item(j).Name & vbCrLf & _
        " число вложенных в нее папок = " & _
        allFolders.Item(i).Folders.Item(j).Folders.Count
  Next
Next
```

Однако понятно, что наращивание числа вложенных циклов для обзора иерархических структур является совершенно бесперспективным занятием. (В нашем примере одна из папок второго уровня — Contacts — имеет также вложенную папку.) Здесь требуется переходить к рекурсивным конструкциям, которые могут выглядеть примерно так:

```
Dim allFolders As Folders
Dim intLevel%  ' номер уровня
intLevel = 0
Set allFolders = Application.GetNamespace("MAPI").Folders
Call FoldersViewRecurse(allFolders, intLevel, "MAPI")

Sub FoldersViewRecurse(allFolders As Folders, intLevel%, strName$)
  Dim i%, FolderName$
  Dim newFolders As Folders
```

```
  ' Вывод информации о папках данного узла иерархической структуры
  Debug.Print "Уровень = "; intLevel; " Узел = "; _
    strName$; Tab(45); " Вложенных папок = "; allFolders.Count
  If allFolders.Count > 0 Then ' есть вложенные папки
    For i = 1 To allFolders.Count  ' обзор вложенных папок
      FolderName$ = allFolders.Item(i).Name
      Set newFolders = allFolders.Item(i).Folders
      ' рекурсивное обращение к самой себе:
      Call FoldersViewRecurse(newFolders, intLevel + 1, FolderName$)
    Next
  End If
End Sub
```

В правильности работы данной конструкции легко убедиться, взглянув на полученную распечатку результатов (рис. 362-2).

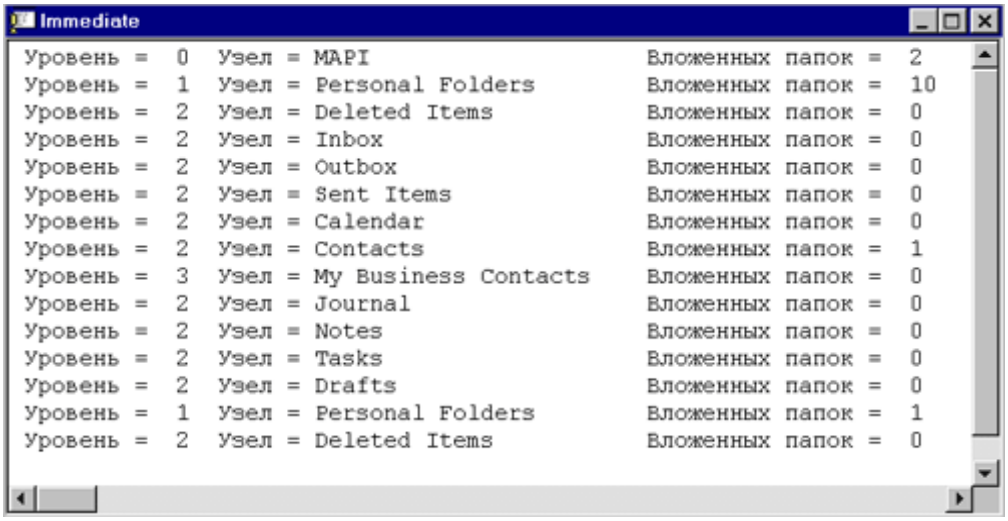


Рис. 362-2

Совет 363. Как добавить новый контакт в папку Outlook

Это производится приблизительно следующим образом:

```
Dim myNewContact As ContactItem
' создание объекта "Контакт"
Set myNewContact = Application.CreateItem(olContactItem)
' далее заполняются нужные поля формы
myNewContact.FirstName = "Андрей"
myNewContact.LastName = "Колесов"
myNewContact.Email1Address = "akolesov@online.ru"
myNewContact.Close olSave    ' сохранить
```

Можно также выдать диалоговое окно "Контакты" для заполнения пользователем:

```
myNewContact.Display
```

Однако данная конструкция записывает новый контакт в стандартную папку "Контакты". Если вам нужно работать с какой-то индивидуальной папкой, вы должны написать такой код (здесь мы вдобавок создаем новую папку):

```
Dim myNewContact As ContactItem
Dim myNewFolder As MAPIFolder
```

```
' Создание папки типа "Контакты"
Set myNewFolder = Application.GetNamespace("MAPI"). _
     GetDefaultFolder(olFolderContacts).Folders.Add("Личная")
' создание объекта "Контакт" для данной папки
Set myNewContact = myNewFolder.Items.Add(olContactItem)
myNewContact.FirstName = "Андрей"
...
```

Совет 364. Как обрабатывать входящие письма

Вам бы хотелось автоматически обрабатывать входящие письма? Это довольно просто сделать с помощью такого кода:

```
Private Sub Application_NewMail()
  ' При поступлении нового письма
  ' производится его обработка
  Dim mailItems As Items
  Dim mailmsg As MailItem

  ' Набор писем из папки "Входящие"
  Set mailItems = Application.Session._
      GetDefaultFolder(olFolderInbox).Items
  Set mailmsg = mailItems.GetLast ' выбираем последнее
  ' далее выполняется анализ письма
  ' (его реквизитов, содержимого и пр.
  ' ...
  ' по результатам анализа можно:
  mailmsg.UnRead = False  ' установить признак "Прочтенное"
  mailmsg.Delete   ' удалить
  mailmsg.Move(myFolder)   ' переместить в другую папку
End Sub
```

Совет 365. Как вставить текст в создаваемое письмо

Это можно сделать, например, с помощью следующей простой макрокоманды, которая создает новое письмо, автоматически заполняет его содержимое и далее предоставляет пользователю возможность вводить остальную информацию и отправлять письмо:

```
Sub NewMailToKolesov()
  ' создание нового письма
  Dim myMail As MailItem
  Set myMail = CreateItem(olMailItem)
  ' заполнение его полей
  myMail.To = "akolesov@online.ru"
  myMail.Subject = "Привет!"
  myMail.body = "Андрей!" & vbCrLf & _
    "Я тут придумал такую классную штуку."
  myMail.Display  ' выводим окно и дополняем текст
End Sub
```

Но возможен и другой вариант — вы уже создали новое письмо и хотите в процессе его ввода сделать вставку какого-то текста. В этом случае вам пригодится макрокоманда такого вида:

```
Sub InsertText()
    'Вставить текст в текущее окно
    Dim myMail As MailItem
    ' выбирает текущее окно (т.е. нового письма)
    Set myMail = Application.ActiveInspector.CurrentItem
```

```
    myMail.body = myMail.body + " Привет семье!"
End Sub
```

Андрей Колесов, Ольга Павлова

© Андрей Колесов, Ольга Павлова, 2001
Авторский вариант. Статья была опубликована с незначительной литературной правкой в журнале "КомпьютерПресс" N 05/2001, CD-ROM.

Совет 366. Как узнать адрес отправителя письма в Outlook 2000

Нам не удалось найти универсальный ответ на этот вопрос. Если, например, пришло новое письмо и мы хотим узнать адрес отправителя, можно написать следующий код в процедуре Application_NewMail():

```
' При поступлении нового письма
' производится его обработка
Dim mailItems As Items
Dim mailmsg As MailItem
Dim Sender$, SenderEmail$

' Набор писем из папки "Входящие"
Set mailItems = Application.Session._
   GetDefaultFolder(olFolderInbox).Items
Set mailmsg = mailItems.GetLast ' выбираем последнее
Sender$ = mailmsg.SenderName
```

В этом случае мы прочитали имя отправителя (в строке From/Откуда). Но как узнать его электронный адрес? К сожалению, подходящего для этой цели свойства мы у объекта MailItem не обнаружили.

А вот если данный отправитель уже внесен в вашу адресную книгу, вы можете узнать его координаты. Это делается следующим образом:

```
Dim repct As Recipient  'описание контакта в книге
' создание объекта с именем отправителя
Set repct = itm.Recipients.Add (mailmsg.SenderName)
recpt.Resolve    'проверка — есть ли какой контакт в книге?
If recpt.Resolved Then  ' есть контакт
  SenderEmail$ = recpt.AddressEntry.address  ' адрес E-mail!
End If
```

Понятно, что имена отправителей в письме и в адресной книге должны быть совершенно идентичны.

Совет 367. Поиск файлов с помощью объекта FileSearch

Задача поиска файлов по некоторым критериям встречается довольно часто. Например, вам нужно найти самый последний измененный файл в некотором каталоге. Для этого можно написать достаточно простой код с использованием функции Dir, с помощью которой делается выборка всех файлов по заданному шаблону:

```
' поиск самого последнего модифицированного файла
Dim FileName$, LastFile$, ThisDate As Date, LastDate As Date
```

```
Dim PathName$, Template$

PathName = "c:\" ' поиск в корневом каталоге C:
Template = PathName & "*.*" ' все файлы
FileName = Dir(Template) ' инициализация
LastDate = #1/1/80#
' просмотр файлов в заданном каталоге
Do While FileName <> ""
  ThisDate = FileDateTime(PathName & FileName) ' дата и время
  ' поиск макс. даты (последней)
  If ThisDate > LastDate Then ' нашли более поздний
    LastDate = ThisDate
    LastFile = PathName & FileName
  End If
  FileName = Dir ' выборка следующего
Loop
If LastFile <> "" Then 'что-то найдено
  MsgBox "Последний по дате файл по шаблону " & _
         Template & vbCrLf & _
         "Имя файла = " & LastFile & vbCrLf & _
         "Дата коррекции = " & LastDate
Else
  MsgBox "Вообще нет файлов с шаблоном " & Template
End If
```

Однако если помимо этого потребуется поиск в подкаталогах, то придется дополнительно сделать их выборку, используя рекурсивные конструкции. В принципе, это не очень сложно (см. совет 230[72]), но все же требует дополнительных усилий и некоторого опыта программирования. Тогда как в среде Office/VBA проблема может быть решена гораздо проще — путем использования объекта FindSearch. Так, например, следующий код позволяет найти все файлы, содержащиеся в каталоге D:\TMP и во всех вложенных подкаталогах:

```
With Application.FileSearch
  .LookIn = "d:\tmp"
  .SearchSubFolders = True
  .FileType = msoFileTypeAllFiles
  If .Execute  > 0 Then
    MsgBox "Число найденных файлов = " & .FoundFiles.Count
    ' вывод имен файлов
    For i = 1 To .FoundFiles.Count
      MsgBox .FoundFiles(i)
    Next i
  Else
    MsgBox "Не найдено подходящих файлов"
  End If
End With
```

Здесь существуют богатые возможности управления режимами выборки, в том числе с поиском по контексту, датам последней модификации и пр. Очень удобно, что имена найденных файлов (свойство FoundFiles) выдаются в виде полных имен. Очевидно, что после получения списка файлов можно выполнить более "тонкую" выборку, например с более жесткими ограничениями по интервалам даты или размера.

Все это можно использовать в обычном VB (или в других системах программирования, поддерживающих ActiveX), подключив библиотеку Microsoft Word 8.0/9.0 Object Library.

[72] http://www.visual.2000.ru/develop/ms-vb/tips/0001-2.htm

Более того, метод Execute позволяет получить список файлов, отсортированный по реквизитам файлов: именам, типам, дате последней модификации и размеру. Так что найти последний измененный файл можно очень просто:

```
With Application.FileSearch
  .LookIn = "d:\tmp"
  .SearchSubFolders = True
  .FileType = msoFileTypeAllFiles
  If .Execute(SortBy:=msoSortByLastModified, _
      SortOrder:=msoSortOrderDescending) > 0 Then
    MsgBox "Последний измененный файл = " & .FoundFiles(i)
  End If
End With
```

Однако тестирование показало, что в Word 2000 сортировка именно по дате модификации почему-то не работает. Но дальнейшие исследования выявили, что она начинает работать (!) после выполнения поиска с сортировкой по размеру файлов. Такой вариант работает корректно:

```
With Application.FileSearch
  .LookIn = "d:\tmp"
  .SearchSubFolders = True
  .FileType = msoFileTypeAllFiles
  .Execute(SortBy:=msoSortBySize)  ' фиктивная выборка, чтобы
                                   ' работала следующая строка кода
  If .Execute(SortBy:=msoSortByLastModified, _
      SortOrder:=msoSortOrderDescending) > 0 Then
    MsgBox "Последний измененный файл = " & .FoundFiles(i)
  End If
End With
```

Совет 368. Сортировка содержимого ListView

Во многих программах, например в Outlook и Windows Explorer, можно выполнять сортировку содержимого элемента управления ListView с помощью щелчка мышью по заголовку колонки. При этом порядок сортировки меняется между вариантами "по возрастанию" и "по уменьшению". Чтобы добавить подобную функциональность в свой проект, создайте в стандартном модуле следующую процедуру:

```
Public Sub SortListView(ByVal lvw As MSComctlLib.ListView, _
   ByVal colHdr As MSComctlLib.ColumnHeader)

  ' установка режима сортировки для указанной колонки
  lvw.SortKey = colHdr.Index - 1
  lvw.Sorted = True

  ' изменение сортировки меняется между
  ' "по возрастанию" и "по уменьшению"
  lvw.SortOrder = 1 Xor lvw.SortOrder
End Sub
```

Чтобы обеспечить выполнение данной операции при щелчке мышью на заголовках, используйте событие ColumnClick для конкретного элемента управления:

```
Private Sub lvwMyListView_ColumnClick(ByVal ColumnHeader As _
  MSComctlLib.ColumnHeader)
  SortListView lvwMyListView, ColumnHeader
End Sub
```

Совет 369. Как определить имя дисковода CD-ROM с помощью FileSystemObject

Как известно, библиотека Scrrun.dll содержит объект FileSystemObject, позволяющий выполнять массу полезных операций с файловой системой. Библиотека входит в состав всех последних модификаций Windows (начиная с обновленного варианта Windows 95) и подключается к проекту посредством команды References; имя библиотеки в списке — Microsoft Scripting Runtime.

С помощью объекта FileSystemObject легко определяется имя дисковода CD-ROM:

```
Dim CDPath as String
Dim fso As New Scripting.FileSystemObject
Dim drv As Drive

For Each drv In fso.Drives  ' перебор всех устройств
  If drv.DriveType = CDRom Then  ' нашли
    CDPath = drv.Path
    Exit For
  End If
Next drv
Set drv = Nothing
Set fso = Nothing
```

Совет 370. Как передать текст из Rich Textbox в Microsoft Word

Задача обработки форматированного текста в VB-приложениях довольно часто решается посредством элемента управления Rich Textbox. В то же время для обработки текстов полезно бывает использовать функции Word (например, проверку грамматики). Соответственно возникает необходимость обмена данными между Rich Textbox и Word.

Это можно сделать, например, с помощью ввода-вывода RTF-файла, но гораздо проще передать информацию через буфер обмена с помощью объекта Clipboard, а затем, используя механизм OLE Automation, открыть приложение Word и вставить в пустой документ отформатированный текст.

Следующая процедура показывает, как выполнить эту операцию (нужно только установить ссылку на библиотеку Microsoft Word 8.0/9.0 Object):

```
Dim wrdApp As Word.Application

Private Sub Form_Load()
  Set wrdApp = New Word.Application
End Sub

Private Sub Command2_Click()
  '
  ' записать текст из Rich Textbox в буфер обмена
  Clipboard.SetText RichTextBox1.TextRTF, vbCFRTF

 ' записать в текст в Word
  With wrdApp
    .Documents.Add    ' новый документ
    .Selection.Paste  ' вставить
      ' запомнить файл
    .ActiveDocument.SaveAs App.Path & "\RTFDOC2.doc", wdFormatDocument
    .Visible = True
```

```
    .Activate  ' сделать документ активным и видимым
  End With
End Sub
```

Совет 371. Как установить VBA SDK

Как известно, для того чтобы обеспечить возможность совместной работы набора VBA SDK (сейчас доступна версия 6.2) со средой VB 6.0, для последнего должен быть установлен Service Pack 3 (подробнее об этой технологии см. статью "Интеграция VBA в бизнес-приложениях независимых разработчиков"[73], КомпьютерПресс 3/2000). В противном случае команда Install Now не сможет инсталлировать мастер VB Integration Wizard.

Однако недавно обнаружилась проблема: несмотря на то что на компьютер был установлен пакет обновления ServicePack 5 (каждый последующий ServicePack автоматически включает все предыдущие обновления), при установке VBA SDK 6.2 выдавалось сообщение об ошибке.

Причину возникновения этой ситуации (судя по всему, это ошибка Microsoft) и решение проблемы нашел С.Новодворский из Брянска. В реестре номер последнего установленного Service Pack записан в параметре latest ключа HKEY_LOCAL_MACHINE\SOFTWARE\MICROSOFT\VISUALSTUDIO\6.0\SERVICEPACKS. Оказывается, перед установкой SDK нужно просто изменить latest на 3, а после инсталляции — восстановить исходное значение.

Совет 372. Как добавить иконку к меню или к кнопке панели инструментов Office

Рассмотрим эту задачу на примере среды Word 97/2000. Как известно, настройка пользовательского интерфейса в пакетах MS Office может выполняться как непосредственно в среде приложения (с помощью диалогового окна "Customize/Настройка"), так и программным образом с применением VBA. В последнем случае можно использовать, например, такую макрокоманду, формирующую новую панель инструментов с кнопкой:

```
Sub AddCommandBarAndButton()
 '
 ' AddCommandBarAndButton Macro
 ' Macro created 21.03.01 by Kolesov Andrei
 '
  Dim myBar As CommandBar
  Dim myControl As CommandBarButton

    ' Создание панели инструментов
  Set myBar = ActiveDocument.CommandBars.Add(Name:="MyNewBar", _
    Position:=msoBarTop, Temporary:=True)
  With myBar
    .Visible = True
    .RowIndex = msoBarRowLast
  End With
    ' Создание кнопки
  Set myControl = myBar.Controls.Add _
   (Type:=msoControlButton, Before:=1)
```

[73] http://www.microsoft.ru/offext/officedev/vba/

```
  With myControl
    .Caption = "Новая_Кнопка"
    .OnAction = "MyNewMacro"
    .FaceId = 16
    .Style = msoButtonIconAndCaption
  End With
End Sub
```

Тут нужно иметь в виду, что при работе с Word панель, формируемая с помощью окна Customize, может храниться в любом документе или шаблоне. В случае программного создания панель сохраняется только в шаблоне Normal.

В приведенном примере мы использовали иконку некоторого встроенного элемента с номером Id = 16 (о проблеме идентификаторов мы поговорим в следующем совете[74]). А как быть, если нужно вставить то или иное собственное изображение?

Здесь может быть предложен такой вариант: создание специальной скрытой панели инструментов, имеющей сугубо вспомогательную функцию — хранение пользовательского набора изображений кнопок. На такую панель (назовем ее IconPanel) можно поместить иконки, отредактированные с помощью встроенного редактора иконок окна Customize. В этом случае создание новой панели инструментов в процессе выполнения VBA-кода будет выглядеть примерно так:

```
' Создание первой кнопки на новой панели инструментов
' копирование созданной ранее кнопки
CommandBars("IconPanel").Controls(1).Copy bar:=myBar, Before:=1
' коррекция параметров кнопки
Set myControl = myBar.Controls(1)
With myControl
  .Caption = "Новая_Кнопка"
  .OnAction = "MyNewMacro"
  .Style = msoButtonIconAndCaption
End With
```

Совет 373. Как определить ID элементов меню и панелей инструментов

Проблема использования встроенных изображений кнопок заключается в том, что их идентификатор ID в явном виде нигде не указан. Однако его можно определить с помощью VBA-кода, указав в явном виде имя панели и кнопки:

```
MsgBox CommandBars("Standard").Controls("New Blank Document").ID
```

Но здесь имеются две проблемы.

Во-первых, на панелях представлены далеко не все реально существующие команды. Чтобы получить названия всех команд, встроенных в данное приложение (их состав и идентификаторы в разных программах могут не совпадать), можно использовать такую процедуру, которая запишет информацию в текстовый файл:

```
Sub OutputControlsID
  Const MaxId% = 4000
  Open "c:"ids.txt" For Output As #1
  ' создаем временную командную панель и включаем
  ' в нее все возможные элементы и кнопки
```

[74] http://www.visual.2000.ru/develop/ms-vb/tips/0105.htm

```
  Set cbr = CommandBars.Add("Временная", msoBarTop, False, True)
  On Error Resume Next ' игнорируем ошибки
                       ' (не всем номерам соответствуют встроенные элементы)
  For I = 1 To MaxID
    cbr.Controls.Add Id: = 1
  Next
  On Error GoTo 0  ' включаем обработку ошибок
  ' записываем идентификатор и название в файл
  For Each btn In cbr.Controls
    Write #1, btn.Id, btn.Caption
  Next
  Cbr.Delete  ' удаляем панель
  Close #1
End Sub
```

Во-вторых, неизвестны номера изображений иконок, представленных в стандартном наборе окна Customize. Тут можно посоветовать просто создать временную кнопку с нужной картинкой и проверить ее код:

```
MsgBox CommandBars("Temporary").Controls("TestIcon").ID
```

Совет 374. Как записать иконку для меню Add-Ins в среде VB

Сказанное выше относится к настройке в среде офисных приложений. А как записать иконку для меню или кнопки, используемой для запуска дополнений Add-in в среде самого VB?

Здесь можно предложить такой вариант. Создайте вспомогательную форму, на которой расположите нужное число элементов управления Image. В каждый элемент поместите иконку в виде растрового изображения 16*16. Кстати, сами иконки можно создавать с помощью того же встроенного редактора окна Customize например пакета Word, копируя их через буфер обмена.

Далее требуется написать следующий код в метод AddToAddInCommandBar класса Connect:

```
Dim cbMenuCommandBar As Office.CommandBarButton
Dim cbMenu As Object
Set cbMenu = VBInstance.CommandBars("Add-Ins")
If Not cbMenu Is Nothing Then ' меню существует
  ' добавляем его к панели инструментов
  Set cbMenuCommandBar = cbMenu.Controls.Add(1)
  CbMenuCommandBar.Caption = "Наш Add-In"
  ' копируем изображение через буфер обмена
  Clipboard.Clear
  Clipboard.SetData = frmAddIns.ImgMenuPic.Picture
  CbMenuCommandBar.PasteFace
  Set AddToAddInCommandBar = cbMenuCommandBar
End If
```

К сожалению, этот метод для VBA не работает, поскольку объект Clipboard имеется только в VB.

Совет 375. Как избавиться от ненужных окон-сообщений

К нам поступил такой вопрос от читателя:

Как написать макрос, чтобы компьютер автоматически отвечал "Нет" на вопросы стандартных диалоговых окон?

Поясню на примере. Скажем, мне надо, чтобы макрос делал что-то в выделенном участке текста Word и не лез в остальной текст. Я записываю макрос стандартным образом (с помощью команды Record New Macro), например прошу найти знаки конца абзаца и заменить их удвоенными знаками конца абзаца. Когда я записываю макрос, то выбираю в стандартном диалоговом окне опцию "заменить все". Затем компьютер сообщает мне, что произвел столько-то замен, и спрашивает, не надо ли провести поиск в оставшемся тексте. Я отвечаю "Нет" и завершаю запись макроса.

Если после этого я выделяю некоторый фрагмент текста и запускаю макрос, то он автоматически заменяет везде в выделенном фрагменте знаки конца абзаца на удвоенные знаки конца абзаца, а затем выводит стандартное окно с предложением произвести замену во всем остальном тексте. Мне это не нужно, так как на самом деле я записываю достаточно большие макросы с разнообразными действиями, и многократно щелкать затем по кнопке "Нет" мне не хочется. Что надо добавить в программу, чтобы избавиться от этой "недоавтоматизации"?

Вот код макроса, о котором шла речь:

```
Sub Макрос1()
 '
 ' замена текста в выделенном фрагменте
  With Selection.Find
    .Text = "^p"
    .Replacement.Text = "^p^p"
    .Forward = True
    .Wrap = wdFindAsk
    .Format = False
  End With
  Selection.Find.Execute Replace:=wdReplaceAll
End Sub
```

Наш ответ:

В данном случае решить проблему достаточно просто — нужно всего лишь сделать установку свойства

```
.Wrap = wdFindStop
```

При этом поиск будет автоматически прекращен по завершении поиска по выделению. Это очень важный момент — программные возможности работы со стандартными диалогами обычно шире по сравнению с составом опций, выдаваемых в окне. Кстати, возможно, будет полезным в конце данного кода записать еще такую строку:

```
Selection.EndKey
```

Это автоматически уберет выделение фрагмента.

А что же делать в случае невозможности управлять свойством Wrap? Здесь пригодился бы более универсальный вариант — программная имитация нажатия нужной клавиши в окне запроса с помощью оператора SendKeys. При этом код макрокоманды выглядел бы следующим образом:

```
Sub Макрос1New() ' ' замена текста в выделенном фрагменте
  With Selection.Find
    .Text = "^p"
    .Replacement.Text = "^p^p"
    .Forward = True
    .Wrap = wdFindAsk
    .Format = False
  End With
  SendKeys "N" ' посылаем в буфер клавиатуры код
      ' клавиши N (горячая клавиша No)
  Selection.Find.Execute Replace:=wdReplaceAll
  Selection.EndKey ' убираем выделение
End Sub
```

Однако нельзя забывать еще один важный момент: далеко не всегда бывает полезно использовать встроенные диалоги Office и точно повторять действия пользователя в среде приложения. В данном случае операция замены одного контекста фрагмента на другой легко производится следующим образом:

```
MyText$ = Chr$(13) ' "конец абзаца"
MyReplacementText = Chr$(13) & Chr$(13)  ' два знака "конец абзаца"
Selection.Text = Replace(Selection.Text, MyText$, MyReplacementText)
```

Совет 376. Используйте XML Parser

В статье "Использование XML DOM в VB"[75] мы рассматривали некоторые возможности взаимодействия с XML-документами. Работа выполнялась с помощью набора объектов библиотеки Microsoft XML 2.0 (MSXML.DLL), которая сейчас обычно называется MS XML Parser.

Это название отражает основное назначение библиотеки (parse — выполнять грамматический разбор), хотя в действительности ее функции выходят за рамки грамматического разбора документа, обеспечивая широкий спектр операций по манипуляциям со структурой и содержимым DOM-документов. Фактически XML Parser предоставляет разработчику приложений механизм создания DOM-документа в виде программного интерфейса взаимодействия с этим документом, а также преобразования его в XML-формат и обратно. В связи с этим хотелось бы сделать небольшое дополнение к упомянутой выше статье.

Одним из основных элементов технологии платформенно-независимого информационного взаимодействия различных приложений является использование объектной модели документов XML (XML Document Object Model, XML DOM), стандарт которой принят комитетом World Wide Web Consortium (W3C). Интерфейс DOM обеспечивает доступ к иерархической структуре, содержимому и стилям документа независимо от платформы и языка программирования.

Следует четко определиться в соотношениях понятий "DOM-документ" и "XML-документ", которые, с одной стороны, почти тождественны, с другой — качественно

[75] http://www.visual.2000.ru/develop/ms-vb/cp0012/xml-doc.htm

различны. DOM-документ, создаваемый приложением, является внутренним объектом последнего, и в общем случае о его физической реализации никому ничего не известно (также как мы работаем с документами Word, ничего не зная о формате его хранения). Содержимое DOM-документа становится доступным для всех остальных приложений путем сохранения его в формате XML-файла. Таким образом, XML-документ является представлением DOM-документа на языке XML.

На примере Visual Basic логика работы с этими документами выглядит следующим образом:

```
Set xmlDoc = New DOMDocument  ' создание нового объекта
' далее - работа по формированию документа
...
xmlDoc.Save "File.xml"    ' сохранение в виде XML-файла
... xmlDoc.Load "NewFile.xml"
' чтение XML-файла
' далее выполняется работа с DOM-объектом
```

Совет 377. Используйте новшества MSXML 3.0

В конце 2000 года Microsoft выпустила новую версию MS XML Parser 3.0 (MSXML3.DLL), призванную заменить MSXML 2.0 и MSXML 2.5, которые поставлялись соответственно в составе Internet Explorer 5.0 и Windows 2000. MSXML 3.0 предоставляет следующие новые функции и возможности по сравнению с версией 2.5:

- полное соответствие W3C-стандартам для технологий Extensible Stylesheet Language Trasformations (XSLT) и XML Path Language (XPath);
- полное соответствие интерфейсу COM/Microsoft ActiveX Simple API for XML (SAX), включая также ряд вспомогательных объектов;
- поддержка безопасного HTTP-доступа (server-safe HTTP access) со стороны серверных приложений;
- ряд улучшений для поддержки DOM и национальных языков;
- высокая степень адаптации к стандартам W3C XML 1.0 и Namespace 1.0, а также к требованиям тестового набора OASIS (Organization for the Advancement of Structural Infomation Standards).

Следует обратить внимание на особенности установки и применения MSXML 3.0. Будучи инсталлирована на компьютер, она не заменяет автоматически предыдущую версию MSXML 2.x — оба варианта библиотеки могут одновременно работать с одним приложением. Например, если некоторое VB-приложение работало с MSXML 2.0, используя следующий код:

```
Dim xml As DOMDocument
Set xmlDoc = New DOMDocument
```

то для переключения на работу с MSXML 3.0 нужно заменить ссылку с MSXML 2.0 на версию 3.0. Однако можно использовать ссылки на обе библиотеки одновременно; в этом случае приведенный выше код будет соответствовать MSXML 2.0, а для работы с MSXML 3.0 потребуется такая конструкция:

```
Dim xml As MSXML2.DOMDocument  ' "2" указывает на стандарт    SAX2
Set xmlDoc = New MSXML2.DOMDocument
```

После установки MSXML 3.0 все компоненты операционной системы (Windows 9x, Windows NT и Windows 2000), в том числе Internet Explorer, продолжают работать с предыдущей версией MSXML 2.x до тех пор, пока не будет выполнена "ручная" замена версий с помощью специальной утилиты XMLINST.EXE.

Загрузить библиотеку MSXML 3.0, набор для разработчика MSXML SDK 3.0 и утилиту XMLINST.EXE можно по адресу www.msdn.microsoft.com/xml[76].

Совет 378. Установка кодировки в MS XML Parser 3.0

Одним из простых, но приятных новшеств новой версии XML Parser является возможность выбора кодировки для записи данных. Ранее использовалась только двухбайтовая кодировка UTF-8, поэтому в обычном текстовом редакторе работать (читать, редактировать) с кириллицей было невозможно.

Теперь же можно использовать привычную Windows-кодировку, указав соответствующий параметр в строке инициализации документа:

```
Dim xmlDoc As DOMDocument
 Set xmlDoc = New DOMDocument
 xmlDoc.loadXML "<?xml version='1.0' encoding='Windows-1251'?>"
```

Андрей Колесов, Ольга Павлова

© Андрей Колесов, Ольга Павлова, 2001
Авторский вариант. Статья была опубликована с незначительной литературной правкой в "КомпьютерПресс" 06/2001, компакт-диск.

Совет 379. Преобразование последовательности байтов в строку шестнадцатеричных символов

Для визуализации информации можно представить последовательность двоичных байтов в виде строки шестнадцатеричных символов (каждый байт представлен парой таких символов). Решение у этой задачи достаточно простое, но следует помнить, что в VB строка может содержать как произвольную последовательность байтов (иначе говоря, символов в однобайтной ANSI-кодировке), так и набор Unicode-символов (этот вариант используется в 32-разрядных версиях VB). В последнем случае каждый символ будет занимать пару последовательных байтов.

Функция преобразования в шестнадцатеричный вид может быть реализована в таком виде:

```
Public Function ConvertBytesToHexString(ByVal strSource$, _
                blnBytes As Boolean) As String
  ' Преобразование последовательности байтов в строку
  ' символов шестнадцатеричного кода
  '
  ' strSource$ - исходная строка байтов
  '   (в вызывающей программе параметр будет неизменным,
  ' так как используется ByVal)
```

[76] http://www.msdn.microsoft.com/xml/

```
  ' blnBytes = True - произвольный набор байтов
  ' = False - символы в кодировке Unicode
  '============================
  Dim lPos&, strChr$, strRtn$
  '
  strRtn = "" 'строка результатов
  If LenB(strSource) > 0 Then
    If (Not blnBytes) Then ' преобразуем Unicode в ANSI
      strSource = StrConv(strSource, vbFromUnicode)
    End If
    ' преобразование
    For lPos = 1 To LenB(strSource)
      strChr = Hex$(AscB(MidB(strSource, lPos, 1)))
       ' добавить "0", чтобы было 2 символа
      If Len(strChr) = 1 Then strChr = "0" & strChr
      strRtn = strRtn & strChr
    Next
  End If
  ConvertBytesToHexString = strRtn
End Function
```

Как видите, функция позволяет работать и с произвольными байтами, и с Unicode-символами. В последнем случае (blnBytes = False) производится автоматическое преобразование из Unicode в ANSI. Следует обратить внимание на то, что такое преобразование переменной не отражается на содержимом исходной строки в вызывающей программе, поскольку используется метод передачи параметра "по значению" (ByVal).

Работа функции ConvertBytesToHexString демонстрируется на следующем тестовом примере:

```
Public Sub Main()
  ' Тестирование процедуры преобразования последовательности
  ' байтов в строку шестнадцатеричных символов
  '
  Dim MySource$, MyResult$
  MySource$ = "Alex Леша"

  ' 1. Как произвольная последовательность байтов
  MsgBox ConvertBytesToHexString(MySource, True)
  ' результат - 41006C006500780020001B04350448043004

  ' 2. Как набор символов в Unicode
  MsgBox ConvertBytesToHexString(MySource, False)
  ' результат - 416C657820CBE5F8E0

  ' 3. Преобразуем в ASNI и выводим как байты
  ' (должен получиться результат, как в п. 2)
  MySource = StrConv(MySource, vbFromUnicode)
  MsgBox ConvertBytesToHexString(MySource, True)
End Sub
```

Сравните полученные результаты, чтобы лучше понять различие Unicode- и ANSI-кодировок. В первом случае каждый символ представлен двумя байтами (байт — двумя шестнадцатеричными символами): первый байт — собственно код символа, а второй — номер кодовой таблицы (для английской он равен 00, для русской — 04). Для английских символов коды Unicode (точнее, его первый байт) и ASNI одинаковы.

Совет 380. Используйте автоматически запускаемые макросы Word

Word содержит несколько специальных имен для макросов, автоматически выполняемых при некоторых предопределенных событиях. Их применение позволяет создать альтернативные варианты обработки вместо встроенных функций. Приведем список этих имен с моментами их выполнения:

```
AutoExec  — при запуске Word или загрузке глобального шаблона
AutoNew   — при создании нового документа
AutoOpen  — при открытии существующего документа
AutoClose — при закрытии документа
AutoExit  — при закрытии Word или выгрузке глобального шаблона
```

Макрос будет выполняться в случаях, если он находится либо в шаблоне Normal, либо в активном документе, либо в шаблоне, на основе которого создан активный документ. Подробнее об использовании автоматически запускаемых макросов см. раздел Auto Macros справочной системы.

Совет 381. Создание строки меню

С помощью диалогового окна Customize (Настройка) можно создавать новые панели инструментов для всех приложений Office 97/2000, но новую строку меню — только в Access. Впрочем, эта задача легко решается с помощью VBA, например в среде Word:

```
Sub CreateNewMenuBar()
 '
 ' программное создание строки меню
    Dim myMenuBar As CommandBar
    Set myMenuBar = CommandBars.Add(Name:="My Menu Bar", _
      Position:=msoBarTop, MenuBar:=True, Temporary:=False)
    With myMenuBar
      .Visible = True
      .RowIndex = msoBarRowLast
    End With
End Sub
```

Выполнив указанный код, мы, однако, обнаружим, что, оказывается, в этом случае новая строка меню не только создается, но и заменяет уже существующую. Точнее, ситуация выглядит следующим образом: в приложении может быть несколько строк меню, но видимым остается только одно. При этом в окне Customize видны все имеющиеся строки меню, но управлять состоянием "видимо/скрыто" (в окошке флажка) здесь нельзя, а доступна лишь операция удаления пользовательской строки меню.

Для управления выводом на экран нужной строки меню можно написать специальную макрокоманду, которая использует диалоговое окно со списком (для вызова макрокоманды лучше создать кнопку на панели инструментов или закрепить на ней комбинацию клавиш):

```
Sub MenuBarVisible()
  ' Управление состоянием строк меню
  UserForm1.Show   ' обращение к форме
End Sub

Private Sub UserForm_Activate()
    Dim cmdb As CommandBar
    Dim nm As String
```

```
    ' формирование списка с созданными строками меню
    For Each cmdb In CommandBars
      nm = cmdb.Name
       ' поиск по имени
      If InStr(1, nm, "menu bar", 1) > 0 Then
         ' найдено меню
         ListBox1.AddItem (nm)
         If cmdb.Visible Then ' строка видимая
           ListBox1.ListIndex = ListBox1.ListCount - 1
         End If
      End If
    Next
End Sub

Private Sub ListBox1_Click()
   ' видимой становится выделенная позиция списка
   CommandBars(ListBox1.List(ListBox1.ListIndex)).Visible = True
End Sub
```

Следует обратить внимание на следующие моменты в приведенном здесь коде:

1. Установка для некоторой строки состояния видимости автоматически скрывает остальные строки.
2. К сожалению, у объекта CommandBar нет свойства, которое позволило бы определить тип объекта (меню, панель и пр.). Поэтому мы вынуждены делать поиск по контексту имени объекта (следовательно, нужно, чтобы имя содержало "Menu Bar").
3. В функции InStr используется текстовый режим поиска (при любом регистре букв). Но в этом случае почему-то обязательно следует указывать первый (необязательный) параметр вызова.

Обратившись к макрокоманде MenuBarVisible, мы получим диалоговое окно со списком строк меню (рис. 381-1). Выбирая элементы списка, мы будем сразу видеть на экране нужную строку меню.

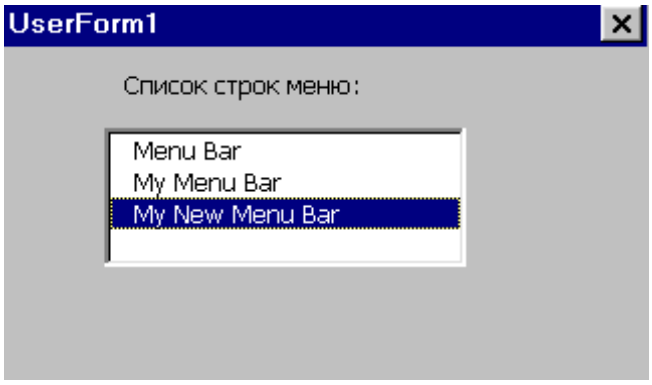


Рис. 381-1

Дальнейшее формирование меню может выглядеть примерно так:

```
Sub CreateNewMenuItem()
    ' формирование меню
    Dim myMenu As CommandBarPopup
    Dim myMenuItem1 As CommandBarPopup
    Dim myMenuItem11 As CommandBarButton
    Dim myMenuItem12 As CommandBarButton
    Dim myMenuItem2 As CommandBarControl
    ' Создаем меню
```

```
    Set myMenu = CommandBars("My Menu Bar").Controls.Add _
      (Type:=msoControlPopup, Before:=1)
    myMenu.Caption = "Новое меню"
    '
    ' Добавляем команды к меню
    '=========================
    ' 1. Это будет ссылка на подменю
    Set myMenuItem1 = myMenu.Controls.Add _
      (Type:=msoControlPopup, Before:=1)
    myMenuItem1.Caption = "Ссылка на подменю"
    ' Формируем подменю
      '1.1. Первая команда
      Set myMenuItem11 = myMenuItem1.Controls.Add _
        (Type:=msoControlButton, Before:=1)
      myMenuItem11.Caption = "Команда 11"
      myMenuItem11.OnAction = "ИмяМакрокоманды11"
      '1.2. Вторая команда
      Set myMenuItem12 = myMenuItem1.Controls.Add _
        (Type:=msoControlButton, Before:=1)
      myMenuItem12.Caption = "Команда 12"
      myMenuItem12.OnAction = "ИмяМакрокоманды12"
    '
    ' 2. Вторая стока меню - команда
    Set myMenuItem2 = myMenu.Controls.Add _
      (Type:=msoControlButton, Before:=1)
    myMenuItem2.Caption = "Команда меню"
    myMenuItem2.OnAction = "ИмяМакрокоманды2"
End Sub
```

В результате у нас получится меню с одной командой и ссылкой на подменю (рис. 381-2).

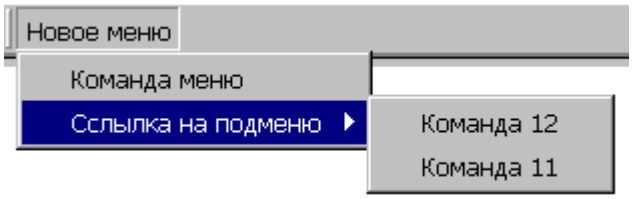


Рис. 381-2

В связи с этим следует иметь в виду, что описания элементов управления CommandBarPopup и CommandBarButton жестко фиксируют его тип, а CommandBarControl позволяет осуществлять динамическое определение типа.

Совет 382. Как узнать значение свойств документа Word

Свойства документа — это такие параметры, которые вы можете увидеть, открыв окно Properties (Свойства). Там находятся два набора свойств — встроенные и пользовательские. Прочитать их программным образом можно с помощью соответственно объектов BuildInDocumentProperties и CustomDocumentProperties, например следующим образом:

```
' Получение свойств документа
  Dim i%, CountOfProperties%
  Dim PropertyValue$
  ' Встроенные
  On Error Resume Next  ' программная обработка ошибок
  CountOfProperties% = ThisDocument.BuiltInDocumentProperties.Count
```

```
  MsgBox "Число встроенных свойств = " & CountOfProperties
  If CountOfProperties > 0 Then
    For i = 1 To CountOfProperties
       Debug.Print ThisDocument.BuiltInDocumentProperties(i).Name;
       PropertyValue = ThisDocument.BuiltInDocumentProperties(i).Value
       If Err.Number <> 0 Then PropertyValue = "Не определено"
       Debug.Print " = " & PropertyValue
       Err.Clear  ' очистка ошибки
    Next
  End If
  ' Пользовательские
  CountOfProperties% = ThisDocument.CustomDocumentProperties.Count
  ' далее идет аналогичная конструкция для объекта CustomDocumentProperties
```

Здесь нужно обратить внимание на необходимость использования программной обработки ошибок. Дело в том, что если какие-то параметры не определены (например, дата последней печати для вновь созданного документа), то обращение к свойству Value вызывает ошибку.

Можно осуществить чтение конкретного свойства документа, в частности количества страниц, указав его индекс или имя (приведенные ниже строки идентичны):

```
ThisDocument.BuiltInDocumentProperties(14).Value
ThisDocument.BuiltInDocumentProperties("Number of pages").Value
```

Однако более правильным является использование встроенных констант VBA:

```
ThisDocument.BuiltInDocumentProperties(wdPropertyPages).Value
```

Это обусловлено тем, что в следующей версии Word, вполне возможно, изменится нумерация и даже названия свойств. Еще раз подчеркнем, что правильное чтение свойств должно выглядеть следующим образом:

```
On Error Resume Next  ' программная обработка ошибок
PropertyValue = ThisDocument.BuiltInDocumentProperties _
    (PropertyName$).Value
If Err.Number <> 0 Then
   ' Свойство не определено
   Err.Clear  ' очистка ошибки
End If
```

Совет 383. Копирование набора данных в XLS-файл с помощью метода CopyFromRecordset

В практике довольно часто встречается задача записи набора данных из БД в лист рабочей книги Excel. Как правило, для этого пишется программа, которая перебирает все столбцы и строки набора данных и переписывает их содержимое в соответствующие ячейки Excel. Однако гораздо проще и быстрее это можно выполнить с помощью малоизвестного метода CopyFromRecordset объекта Range. Вот пример выполнения этой операции, реализованной в виде макрокоманды Excel:

```
Sub CopyRecordset()
  '
  ' Копирование набора данных в рабочую книгу
  Dim db As DAO.Database
  Dim rs As DAO.Recordset
  '  открываем набор данных
```

```
  Set db = DAO.OpenDatabase("C:\vb-db\xmltest.mdb")
  Set rs = db.OpenRecordset("SELECT * From Employees")

  Call CopyRecordsetToWorkbook(rs)  ' копирование

  rs.Close  ' закрываем
  db.Close
  Set rs = Nothing  ' освобождаем
  Set db = Nothing
End Sub

Sub CopyRecordsetToWorkbook(rs As DAO.Recordset)
   '  Копирование набор в ячейки листа
   ThisWorkbook.Worksheets(1).Range("A1").CopyFromRecordset rs
End Sub
```

Если вы хотите сделать такое преобразование из другого приложения, написанного, например, на VB, то можно воспользоваться механизмом OLE Automation. В этом случае пригодится приведенная выше процедура CopyRecordset, а подпрограмма CopyRecordsetToWorkbook будет выглядеть так:

```
Sub CopyRecordsetToWorkbook(rs As DAO.Recordset)
   Dim xlApp As Excel.Application
   Dim xlBook As Excel.Workbook
   '
   ' открываем приложение и книгу
   Set xlApp = New Excel.Application
   Set xlBook = xlApp.Workbooks.Open("d:\tmp\book2.xls")
      ' копирование
   xlBook.Worksheets(1).Range("A1").CopyFromRecordset    rs
   'сохраняем, закрываем, очищаем
   xlBook.Save
   xlBook.Close False
   xlApp.Quit
   Set xlBook = Nothing
   Set xlApp = Nothing
End Sub
```

Имейте в виду, что Excel 97 поддерживает работу метода CopyFromRecordset только для простых наборов данных DAO (ODBCDirect-наборы не поддерживаются), однако Excel 2000 может использовать все варианты наборов данных DAO и ADO.

Совет 384. Использование Script Control в VBA

В совете 269[77] мы говорили об элементе управления Script Control 1.0 (его можно скачать по адресу www.msdn.microsoft.com/scripting[78]), с помощью которого можно вычислять математические выражения, заданные в виде символьной строки. Например, следующим образом:

```
a$ = "(2 * 5) + 3"
Result = ScriptControl.Eval(a$)
```

Один из наших читателей обнаружил, что по какой-то причине этот элемент управления "не хочет" работать в среде VBA. Действительно, при попытке переместить его с панели

[77] http://www.visual.2000.ru/develop/ms-vb/tips/0004-2.htm
[78] http://www.msdn.microsoft.com/scripting

инструментов на форму выдается сообщение об ошибке: "Unexpected call to method or property access".

Почему такое происходит, мы понять не смогли, но нашли альтернативное решение вопроса: можно вообще обойтись без применения элемента управления, работая напрямую с объектом. Для этого нужно посредством команды Tools|Reference подключить библиотеку Microsoft Srcipt Control 1.0 и использовать такой код, обращаясь непосредственно к объекту:

```
Sub MyScript()

  ' описание и создание объекта
  ' (командой Reference подключите библиотеку Microsoft Script Control)
  Dim myScriptContol As MSScriptControl.ScriptControl
  Set myScriptContol = New MSScriptControl.ScriptControl

  ' обязательно нужно определить языковой механизм
  myScriptContol.Language = "vbscript"

  ' далее работаете с объектом
  MsgBox myScriptContol.Eval("(2+5)* 6")
End Sub
```

Совет 385. На что нужно обратить внимание при замере интервалов времени

В одной из телеконференций мы нашли вопрос об "аномальном" поведении функции Timer. В документации сказано, что функция возвращает значение текущего времени суток в секундах (в виде величины типа Single), из чего со всей очевидностью следует, что она должна монотонно возрастать (но в полночь отсчет опять начнется с нуля!). Однако задавший этот вопрос обнаружил, что иногда значение переменной sngDiff в приведенном ниже фрагменте кода, может оказаться отрицательным.

```
Dim sngPrev As Single, sngDiff As Single
 sngPrev = Timer()
 sngDiff = Timer() - sngPrev
```

Чтобы убедиться в этом, достаточно выполнить такой тестовый пример:

```
Dim sngPrev As Single, sngDiff As Single
Dim lngIndex As Long

For lngIndex = 1 To 10000
  sngPrev = Timer
  sngDiff = Timer - sngPrev
  If sngDiff < 0 Then
    MsgBox "Время пошло вспять = " & sngDiff
  End If
Next
```

Интересно, что если написать вычисление промежутку времени как:

```
sngDiff = CSng(Timer) - sngPrev
```

то все начинает работать как следует.

В чем же здесь загвоздка? Ситуация действительно кажется странной, но только на первый взгляд. Дело в том, что функция Timer работает на уровне точности секунд и не годится для обработки тысячных долей секунды. При выводе значения Timer вы увидите число лишь с двумя знаками после запятой.

Очевидно, что при преобразовании времени из какого-то внутреннего формата происходит некоторая потеря точности. Следовательно, надо знать организацию самой функции Timer, но вполне вероятно, что она имеет собственный формат данных. Поэтому возможно, что при выполнении:

```
Timer - sngPrev
```

последняя переменная преобразуется из Single во внутренний формат Timer, и в этот момент происходит потеря точности в последнем разряде. В общем, это достаточно обычное явление, когда вы занимаетесь преобразованием форматов числовых данных на грани точности. Попробуйте выполнить такой пример:

```
A = 1.4
B = A
C = B + 0.2
If C < A Then MsgBox ("Ну, дела!")
```

Казалось бы, получение сообщения "Ну, дела!" в принципе невозможно. Однако оно будет появляться всякий раз, если вы сделаете такое определение типов данных:

```
Dim a As Single, c As Single, b As Integer
```

Это означает, что в данном случае будет происходить изменение значения числа при преобразовании его из вещественного формата в целочисленный (это мы к тому, что точный внутренний формат Timer нам не известен).

Итак, для указанных промежутков времени Timer просто не подходит. Что же делать?

В совете 193[79] для более точного измерения промежутков времени мы рекомендовали использовать функцию GetTickCount:

```
Declare Function GetTickCount& Lib "kernel32"
```

Она выдает время в миллисекундах с момента последнего старта или перезагрузки операционной системы (то есть обнуление счетчика будет происходить только через 49 суток непрерывной работы компьютера). Обратите внимание, что даже для наносекундных интервалов время не пойдет в обратную сторону, так как мы имеем дело с уже преобразованными целочисленными значениями.

Совет 386. Не применяйте неявного преобразования данных

Мы уже несколько раз повторяли этот совет и сейчас просто хотим привести еще одно подтверждение его правильности.

Порой в программах приходится встречать такие логические конструкции:

[79] http://www.visual.2000.ru/develop/ms-vb/tips/9906.htm

```
If Len(SomeText$) Then
  ' выполнение условия, если длина строки ненулевая
```

или:

```
If Err.Number Then  ' Если код ошибки ненулевой
```

В данном случае этот код работает правильно, но потенциально угрожает надежной (прогнозируемой) работе программы. Угроза заключается в том, что после ключевого слова If должно идти логическое условие. Поэтому строгий синтаксис языка должен допускать наличие кода только следующего вида:

```
If Len(SomeText$) > 0 Then
```

Однако, к сожалению, VB позволяется использовать и такую запись:

```
If Len(SomeText$) Then
```

Эта запись на самом деле эквивалентна нижеприведенной синтаксически правильной конструкции:

```
Dim blnValue As Boolean
blnValue = Len(SomeText$)
If blnValue Then
```

Работоспособность этого кода определяется тем, что любое ненулевое целое число преобразуется в булево значение True и, следовательно, последующая проверка срабатывает верно. Итак, в чем же состоит опасность для надежности программы?

Предположим, что теперь мы хотим модифицировать код таким образом, чтобы некоторые действия выполнялись при пустом значении строки. Казалось бы, для этого нужно просто заменить код условия, используя логическое отрицание:

```
If Not Len(SomeText$) Then MsgBox "Пустое значение"
```

Но здесь вы обнаружите неприятную неожиданность: сообщение "Пустое значение" будет выдаваться всегда — при любом значении строковой переменной. Действительно, приведенные ниже строки кода эквивалентны:

```
If Not Len("") Then
If Not 0 Then
If -1 Then
If True Then  ' выполняется код
```

И эти строки также эквивалентны:

```
If Not Len("КуКу") Then
If Not 4 Then
If -5 Then
If True Then  ' выполняется код
```

Такое происходит потому, что сначала выполняется логическая инверсия целой переменной, которая лишь потом преобразуется в логического значение. Чтобы не мучиться с подобными проблемами, записывайте логические выражения только в явном виде. При этом и логика программы выглядит гораздо понятнее:

```
If Len(SomeText$) > 0 Then ...
```

Совет 387. Передача XML-данных в виде атрибутов

В статье "Использование XML DOM в VB"[80] при передаче данных использовались тэги (элементы) языка XML. Но довольно часто удобнее бывает применять для вывода атрибуты элементов. Для сравнения двух вариантов внимательно посмотрите на код, приведенный в листинге 387[81].

А теперь взгляните на полученный результат (рис. 387).

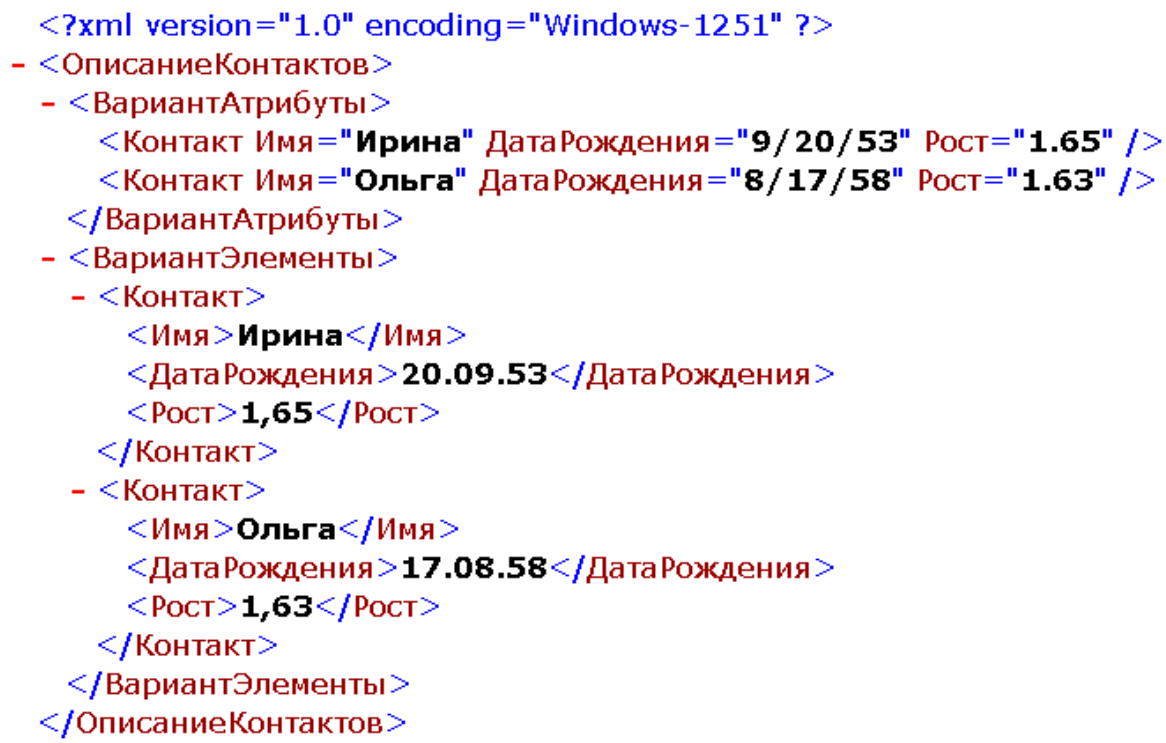


Рис. 387

При всей схожести мы видим одно принципиальное различие: значения атрибутов записаны в формате международных региональных установок (независимо от установок конкретной операционной системы), а элементов — в формате национальных установок данного компьютера. Это означает, что нас ожидают определенные проблемы при передачи информации между системами с различными региональными установками.

Вот как выглядит код чтения записанного ранее XML-файла:

```
Public Sub XMLinput()
   ' Чтение XML-файла
  Dim CommNode As IXMLDOMElement
  Dim curNode As IXMLDOMElement
  Dim subNode As IXMLDOMElement
  Dim FullName$, Birthdate As Date, Height As Single
```

[80] http://www.visual.2000.ru/develop/ms-vb/cp0012/xml-doc.htm
[81] http://www.visual.2000.ru/develop/ms-vb/tips/0106.htm

```
  Set xmlDoc = New DOMDocument
  xmlDoc.Load xmlFile$

  ' чтение данных
  ' Чтение узла "ВариантАтрибуты"
  Set CommNode = xmlDoc.selectSingleNode("//ВариантАтрибуты")
  For Each curNode In CommNode.selectNodes("Контакт")
    FullName$ = curNode.getAttribute("Имя")
    Birthdate = curNode.getAttribute("ДатаРождения")
    Height = Val(curNode.getAttribute("Рост"))
    MsgBox FullName & " " & Birthdate & " " & Height
  Next

  ' Чтение узла "ВариантЭлементы"
  Set CommNode = xmlDoc.selectSingleNode("//ВариантЭлементы")
    For Each curNode In CommNode.selectNodes("Контакт")
    FullName$ = curNode.selectSingleNode("Имя").Text
    Birthdate = curNode.selectSingleNode("ДатаРождения").Text
    Height = curNode.selectSingleNode("Рост").Text
    MsgBox FullName & " " & Birthdate & " " & Height
  Next
End Sub
```

В обоих случаях мы вроде бы получаем одинаково правильные результаты. Но есть один важный нюанс: первый вариант работает с международным форматом данных и поэтому не зависит от региональных установок, а второй будет работать только при условии, что региональные установки источника и приемника информации совпадают.

Листинг 387. Вывод XML-данных в виде элементов и атрибутов

```
Dim xmlDoc As DOMDocument

Public Sub Main()

  ' Создание экземпляра объекта
  ' xmlDoc - переменная уровня модуля!
  Set xmlDoc = New DOMDocument

  ' записываем XML-константу объекта
  xmlDoc.loadXML "<?xml version='1.0'" & _
    "encoding='Windows-1251'?>" & _
    "<ОписаниеКонтактов/>"

  ' Создание узла "ВариантАтрибуты"
  Call CreateNewNode("ВариантАтрибуты", True)

  ' Создание узла "ВариантАтрибуты"
  Call CreateNewNode("ВариантЭлементы", False)
  '
  xmlDoc.Save "d:\file2.xml"
End Sub

Public Sub AddContactInfomation( _
  varAttrNode As IXMLDOMElement, FirstName$, _
  Birthdate As Date, Height As Single, VarType As Boolean)

  Dim xmlField As IXMLDOMElement
  '
  ' создание нового подчиненного узла
  Set xmlField = varAttrNode.appendChild _
```

```
    (xmlDoc.createElement("Контакт"))
  ' запись информации
  Call addData(xmlField, "Имя", FirstName$, VarType)
  Call addData(xmlField, "ДатаРождения", Birthdate, VarType)
  Call addData(xmlField, "Рост", Height, VarType)
End Sub


Public Sub addData(xmlField As IXMLDOMElement, _
    dName$, dValue As Variant, VarType As Boolean)
  ' добавить атрибут для узла
  Dim attr As IXMLDOMAttribute
  Dim xmlData As IXMLDOMElement
  '
  If VarType Then ' "ВариантАтрибуты"

    Set attr = xmlDoc.createAttribute(dName)
    attr.Value = dValue
    xmlField.Attributes.setNamedItem attr
  Else
    Set xmlData = xmlField.appendChild _
        (xmlDoc.createElement(dName))
    xmlData.Text = dValue
  End If
End Sub

Public Sub CreateNewNode(NodeName$, VarType As Boolean)

  ' Создание одного узла и запись в него информации
  ' VarType = True (в виде атрибутов)
  ' = False (в виде элементов)
  Dim NewNode As IXMLDOMElement

  ' Создание узла "ВариантЭлементы"
  Set NewNode = xmlDoc.documentElement.appendChild _
                (xmlDoc.createElement(NodeName))
  ' запись информации в виде атрибутов
  Call AddContactInfomation(NewNode, "Ирина", _
                "20.09.1953", 1.65, VarType)
  Call AddContactInfomation(NewNode, "Ольга", _
                "17.08.1958", 1.63, VarType)
End Sub
```

Андрей Колесов, Ольга Павлова

© Андрей Колесов, Ольга Павлова, 2001
Авторский вариант. Статья была опубликована с незначительной литературной правкой в "КомпьютерПресс" 09/2001, компакт-диск.

Совет 388. Передача адреса процедуры через структуру

Как известно, функция AddressOf позволяет передавать в качестве параметра адрес VB-процедуры при обращении к DLL-процедурам вообще и к Win API в частности. Обычно это нужно для реализации механизма "обратного вызова" (Callback). Однако порой бывает необходимо передавать такой адрес в виде одного из полей структуры данных. Для решения подобной задачи следует иметь в виду, что ключевое слово AddressOf можно

использовать и при обращении к VB-процедуре, в связи с чем можно предложить такой вариант программы:

```
Public Sub Main()
   Type MySturture
     lpfnProc As Long  ' адрес процедуры
   End Type
   Dim MyStruc
    ' вычисление адреса процедуры MyProc
   MyStruc.lpfnProc = FcnPtr(AddressOf MyProc)
End Sub

Public Function FcnPtr(ByVal Whatever As Long) As Long
  ' фиксируем значение переданного адреса процедуры
  FcnPtr = Whatever
End Function

Public Sub MyProc()
  ' ...
End Sub
```

Совет 389. Программная регистрация ActiveX DLL и OCX

Обычно регистрация (или ее отмена) ActiveX-компонентов выполняется с помощью автономной утилиты regsvr32.exe. Если необходимо выполнять процедуры регистрации в момент выполнения вашего VB-приложения, то можно воспользоваться обращением к этой утилите с помощью Shell. Однако существует еще один способ проведения таких операций, недостатком которого является необходимость "железного" включения имени нужного компонента в код программы.

Дело в том, что каждый ActiveX-компонент (ActiveX DLL или OCX) имеет функции DllRegisterServer и DllUnregisterServer, выполняющие операции регистрации/отмены регистрации над собственным компонентом. И обратиться к ним можно напрямую, как к обычной DLL-функции.

Например, если вы хотите программно проводить операции регистрации компонента COMCTL32.OCX, то в программе нужно описать две такие функции:

```
' функция регистрации компонента COMCTL32.OCX
Declare Function RegComCtl32 Lib "COMCTL32.OCX" Alias _
    DllRegisterServer() As Long
' функция отмены регистрации компонента COMCTL32.OCX
Declare Function UnRegComCtl32 Lib "COMCTL32.OCX" Alias _
    DllUnregisterServer() As Long
```

Однако следует иметь в виду, что если вы не указали полный путь к файлу, то его поиск будет осуществляться только в системном или текущем каталоге. Кроме того, при выполнении операций целесообразно реализовать механизм анализа возможных ошибок.

Приведем пример кода регистрации библиотеки Test.DLL, которая хранится в произвольном каталоге C:\MyApp:

```
Declare Function RegTestDLL Lib "Test.DLL" Alias _
    DllRegisterServer() As Long
Const ERROR_SUCCESS = 0&
Dim retCode As Long
On Error Resume Next  ' включаем программную обработку ошибок
```

```
ChDrive "C:"         ' Устанавливаем нужный
ChDir "C:\MyApp"     ' каталог текущим
regCode = RegTestDLL()  ' регистрация Test.DLL
' анализ возможных ошибок
If Err <> 0 Then
  MsgBox "Файл Test.DLL не найден"
Else
  If regCode <> ERROR_SUCCES Then
    MsgBox "Операция регистрации не выполнена"
  End If
End If
```

Совет 390. Быстрое перемещение между процедурами

В окне кода VB передвигаться между процедурами модуля или формы можно с помощью быстрых клавиш — Ctrl + Page Down и Ctrl + Page Up.

Совет 391. Как посчитать число строк в текстовом окне

Если вы определили текстовое поле как "многостроковое", может возникнуть необходимость узнать число строк. Это можно сделать с помощью такого простого кода:

```
Private Sub Command1_Click()
  Dim myParas As Variant
  myParas = Split(Text1.Text, vbNewLine)
  MsgBox "Число строк = " & (UBound(myParas) + 1)
End Sub
```

Но данный вариант позволит получить только число строк, разделенных "жестким образом" (возврат каретки). Для того чтобы получить фактическое число строк в данном текстовом окне, можно воспользоваться обращением к API-функции SendMessageAsLong:

```
Private Declare Function SendMessageAsLong Lib "user32" _
  Alias "SendMessageA" (ByVal hWnd As Long, ByVal wMsg As Long, _
  ByVal wParam As Long, ByVal lParam As Long) As Long
Const EM_GETLINECOUNT = 186
Private Sub Command2_Click()
  Dim lCount As Long
  lCount = SendMessageAsLong(Text1.hWnd, EM_GETLINECOUNT, 0, 0)
  MsgBox "Фактическое число строк = " & lCount
End Sub
```

Совет 392. Как определить позицию курсора в Rich Textbox

Если в качестве текстового редактора вы используете элемент управления Rich Textbox, то полезно узнать не только число строк (о чем говорилось в предыдущем совете), но также, например, и текущую позицию курсора. Это можно сделать с помощью еще одной API-функции — SendMessageByNum:

```
Private Declare Function SendMessageByNum Lib "user32" _
 Alias "SendMessageA" (ByVal hwnd As Long, ByVal wMsg As Long, _
 ByVal wParam As Long, ByVal lParam As Long) As Long
Private Const EM_LINEFROMCHAR = &HC9 Private Const EM_LINEINDEX = &HBB
Public Function GetCurrentLine(TxtBox As Object) As Long
  ' определение текущей строки в окне
  With TxtBox
    GetCurrentLine = SendMessageByNum(.hwnd, _
```

```
        EM_LINEFROMCHAR, CLng(.SelStart), 0&) + 1
  End With
End Function
Public Function GetCurrentColumn(TxtBox As Object) As Long
  ' определение текущей колонки в окне
  With TxtBox
    GetCurrentColumn = .SelStart - SendMessageByNum(.hwnd, _
      EM_LINEINDEX, -1&, 0&) + 1
  End With
End Function
```

Вот как их можно использовать:

```
Private Sub Command1_Click()
  MsgBox "Текущая строка = " & GetCurrentLine(RichTextBox1)
End Sub
Private Sub Command2_Click()
  MsgBox "Текущая колонка = " & GetCurrentColumn(RichTextBox1)
End Sub
```

Совет 393. Программная имитация нажатия кнопок

Предположим, на вашей форме Form1 расположены несколько командных кнопок и вам нужно выполнить программную имитацию их нажатия:

```
Private Sub Command1_Click()
  MsgBox "Кнопка 1 нажата!"
End Sub
Private Sub Command2_Click()
  MsgBox "Кнопка 2 нажата!"
End Sub
```

К примеру, на другой форме Form2 имеется кнопка AllCommands, щелчок которой должен вызывать нажатие двух кнопок на Form1. Обратиться непосредственно к процедурам CommandN_Click нельзя, так как они имеют статус Private. Для решения этой проблемы следует воспользоваться свойством Value, которое управляет состоянием кнопки:

```
Private Sub AllCommands_Click()
  Form1.Command1.Value = True
  Form1.Command2.Value = True
End Sub
```

Установка значения Command.Value = True автоматически вызывает выполнение события Click соответствующей кнопки.

Совет 394. Динамическое управление заголовками колонок DataGrid

При программной установке источника данных элемента управления DataGrid нельзя указывать имена заголовков колонок в режиме проектирования. Поэтому DataGrid применяет имена колонок самого набора данных, что не всегда бывает удобным для работы.

Эта проблема решается элементарно: в SQL-операторе нужно просто указать альтернативные имена полей. Например, вместо

```
SELECT pub_id, pub_name FROM pubs
```

написать

```
SELECT pub_id AS Учетный_Номер, pub_name AS Издатель FROM pubs
```

В результате DataGrid выдаст на экран в качестве заголовков имена "Учетный_Номер" и "Издатель" вместо соответствующих идентификаторов pub_id и pub_name.

Совет 395. А вы знаете о свойстве LockControls формы?

Недавно при работе со считанным из Интернета небольшим программным примером я столкнулся со странной ситуацией. Для удобства работы я хотел немного уменьшить форму проекта и для этого немного передвинуть расположенные на ней элементы управления. Но не тут-то было — компоненты выделялись, но передвигаться или менять размеры не хотели. При этом выделяемые элементы обрамлялись не черными квадратиками, как обычно, а белыми, показывая тем самым, что никакие перемещения не разрешены.

Оказалось, что такой режим блокировки элементов управления формы задается свойством LockControls = -1 (True), нигде не описанным и не упомянутым (кстати, в электронной документации вообще нет четкого описания свойств формы). Получается, что остановка/отмена LockControls может выполняться только непосредственным редактированием FRM-файла, которое будет выглядеть приблизительно следующим образом:

```
Begin VB.Form MyForm
   Caption         =   "Форма с блокировкой компонентов"
   ...
   LockControls    =   -1  'True
   ...
```

Совет 396. Используйте функцию ExtFloodFill для цветной заливки поверхности

Те, кто работал с QuickBasic (в среде DOS), возможно, помнят, что там был оператор PAINT, который позволял заливать поверхности фигур произвольных очертаний. В системе VB этот оператор отсутствует и встроенные VB-функции позволяют выполнять заливку только "стандартных" фигур, например прямоугольника или круга, что явно недостаточно для решения многих графических задач.

Тем не менее выход существует — нужно использовать функцию ExtFloodFill из состава Win32 GDI API, которая имеет следующее описание:

```
Private Declare Function ExtFloodFill Lib "gdi32" _
  (ByVal hDC As Long, ByVal X As Long, ByVal Y As Long, _
  ByVal crColor As Long, ByVal wFillType As Long) As Long
```

Параметр hDC — номер описателя объекта. В данном случае допустимым объектом является форма (Form) или элемент управления PictureBox, номер описателя определяется с помощью их свойства hDC. Цветовая фактура заливки задается с помощью двух свойств объекта (их нужно установить перед обращением к ExtFloodFill) — FillColor (цвет) и FillStyle (тип фактуры).

X и Y — координаты точки, из которой выполняется заливка. Однако следует иметь в виду, что они должны быть заданы в пикселах, а не в логических координатах, которые могут быть установлены свойством ScaleMode (по умолчанию Twip).

crColor и wFillType — взаимосвязанные параметры, определяющие режим выбора площади заливки:

- wFillType = 0 (FLOODFILLBORDER) — заливка выполняется в пределах фигуры, ограниченной контуром цвета crColor. Понятно, что цвет "начальной" точки заливки не может быть равным crColor.
- wFillType = 1 (FLOODFILLSURFACE) — заливка выполняется в точках площади, которые имеют цвет crColor. В этом случае, наоборот, цвет "начальной" точки заливки должен быть равным crColor.

В составе GUI API также предусмотрена функция FloodFill, которая является частным случаем ExtFloodFill (случай FLOODFILLBORDER).

Следует обратить внимание на один момент, не отмеченный в документации и обнаруженный нами в ходе эксперимента. Оказалось, что функция ExtFloodFill срабатывает, только если перед обращением к ней выполнено обращение к свойству object.Point (x, y). Впрочем, такую операцию выполнять в любом случае полезно, поскольку цвет "начальной" точки понадобится для установки параметра crColor (FLOODFILLSURFACE) или, напротив, для проверки значения этого же параметра (FLOODFILLBORDER).

На листинге 396-1[82] приведен пример процедуры Paint, которая имитирует оператор PAINT системы QB. Обратите внимание, что параметр BorderColor при обращении к ней нужно задавать только для случая FLOODFILLSURFACE — в другом режиме он определяется автоматически внутри процедуры с помощью Point (x, y).

На рис. 396-1 и 396-2 показаны возможности применения функции ExtFloodFill соответственно в режимах FLOODFILLBORDER и FLOODFILLSURFACE.

[82] http://www.visual.2000.ru/develop/ms-vb/tips/0109.htm

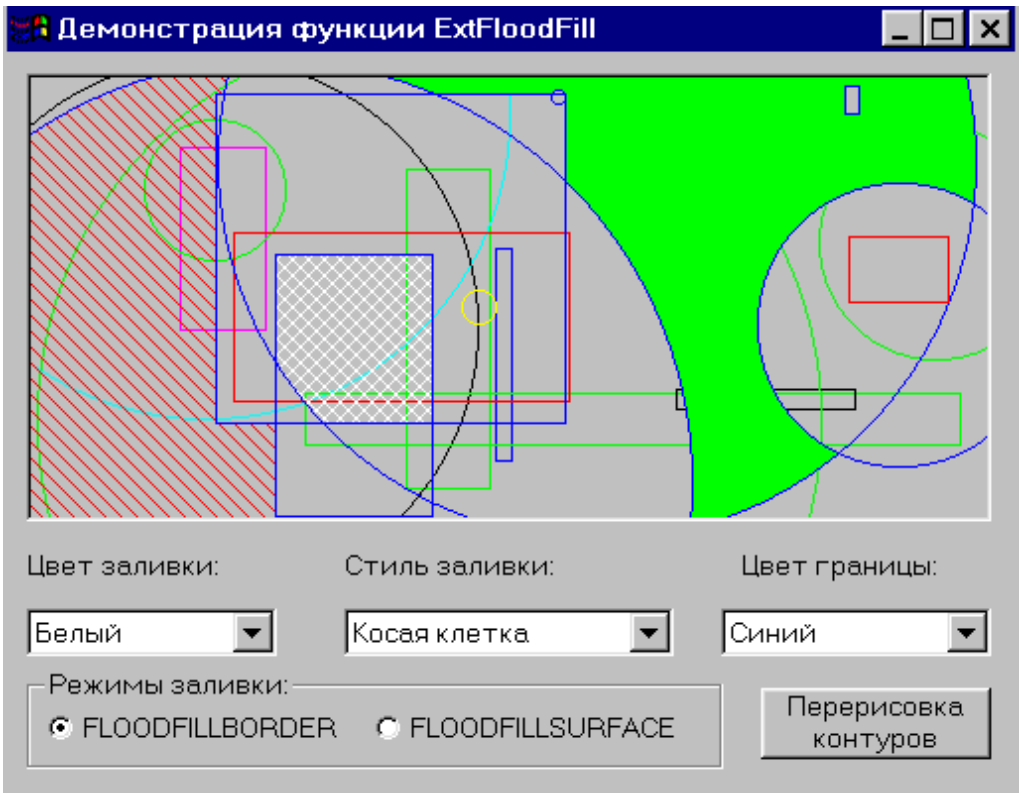


Рис. 396-1

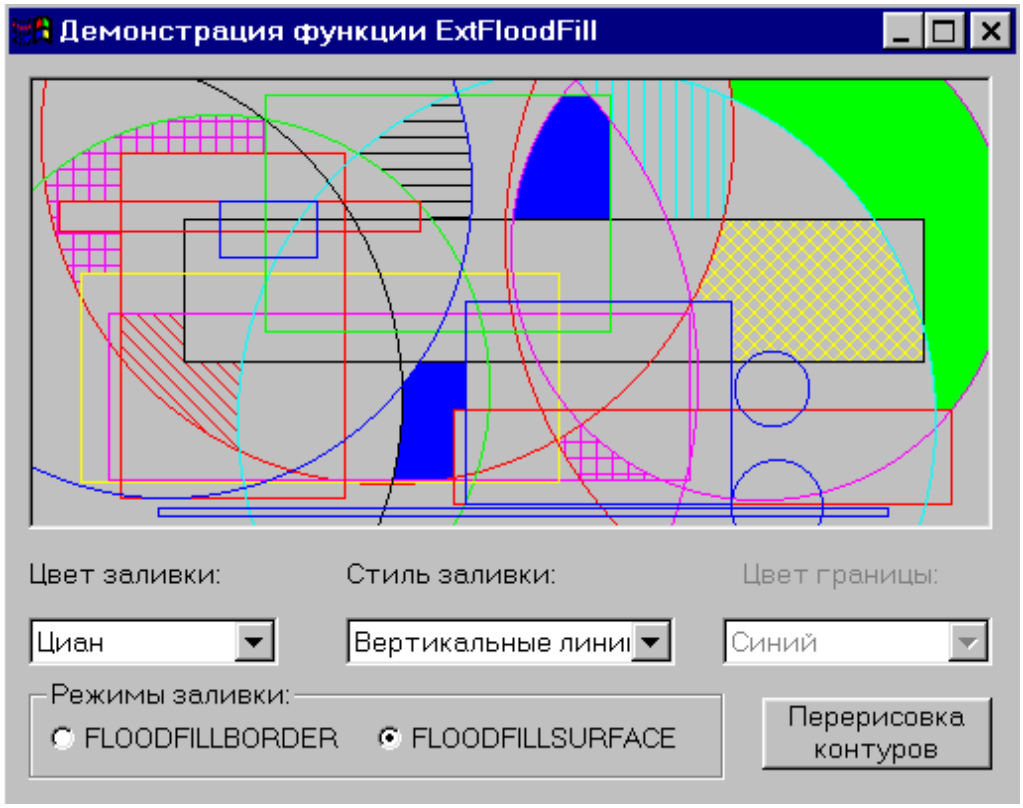


Рис. 396-2

А в листингах 396-2 и [83]396-3[84] приведен код формы Flood.frm и модуля AddProc.bas, которые используются в этом демонстрационном примере. Заливка выполняется в элементе управления PictureBox.

Обратите также внимание, что при рисовании начальных контуров (функция DrawShapes) последние пять линий изображаются фиксированным, синим цветом. Это сделано специально для демонстрации работы режима FLOODFILLBORDER. Ведь для этого случая граница не должна иметь "дырок", которые получаются при наложении линий других цветов. Таким образом, наш пример устойчиво будет работать только для случая границы синего цвета. Это вполне соответствует реальной ситуации — перед заливкой необходимо сделать контур нужного цвета.

Листинг 396-1. Модуль FloodFll.bas — процедура Paint для заливки фигуры произвольных очертаний

```
Option Explicit
 Private Declare Function ExtFloodFill Lib "gdi32" _
   (ByVal hDC As Long, ByVal X As Long, ByVal Y As Long, _
   ByVal crColor As Long, ByVal wFillType As Long) As Long
 ' Значения режимов заливки

Public Enum FillModes
   FLOODFILLBORDER = 0
   FLOODFILLSURFACE = 1
End Enum

Public Sub Paint(ByVal Canvas As Object, ByVal X As Single, _
    ByVal Y As Single, ByVal FillColor As Long, _
     Optional ByVal FillStyle As FillStyleConstants = vbFSSolid, _
     Optional ByVal BorderColor As Long = vbBlack, _
     Optional ByVal Flags As FillModes = FLOODFILLBORDER)
   ' Заливка поверхности фигуры произвольных очертаний
   Dim xP As Long, yP As Long
   Dim oldFillColor As Long
   Dim oldFillStyle As Long
   ' Работает только с Form и Picture
   If Not TypeOf Canvas Is Form Then
      If Not TypeOf Canvas Is PictureBox Then
        MsgBox "Контур должен быть формой или картинкой"
        Exit Sub
      End If
   End If
   Dim PointColor As Long
   ' почему-то нужно обязательно выполнить операцию Point
   PointColor = Canvas.Point(X, Y)
   '
   ' Впрочем, это пригодится для установки режима заливки
   ' и проверки параметров
   If Flags = FLOODFILLBORDER Then
     If PointColor = BorderColor Then
       MsgBox "Ошибка: совпадают цвета 'начальной'" & _
         "точки и границы в режиме FLOODFILLBORDER"
       Exit Sub
     End If
   Else  ' в этом режиме нужно установить текущий цвет
     BorderColor = PointColor
```

[83] http://www.visual.2000.ru/develop/ms-vb/tips/0109.htm
[84] http://www.visual.2000.ru/develop/ms-vb/tips/0109.htm

```
   End If
   ' Преобразование координат (логические данного объекта) в пикселы
   xP = Canvas.ScaleX(X, Canvas.ScaleMode, vbPixels)
   yP = Canvas.ScaleY(Y, Canvas.ScaleMode, vbPixels)
   ' Сохранение текущих атрибутов и установка новых
   oldFillColor = Canvas.FillColor
   oldFillStyle = Canvas.FillStyle
   Canvas.FillColor = FillColor
   Canvas.FillStyle = FillStyle
   ' Заливка !
   Call ExtFloodFill(Canvas.hDC, xP, yP, BorderColor, Flags)
   ' Восстановление значений атрибутов
   Canvas.FillColor = oldFillColor
   Canvas.FillStyle = oldFillStyle
End Sub
```

Листинг 396-2. Модуль Flood.frm — демонстрационный пример практического использования процедуры Paint

```
Option Explicit
 Private m_X As Single
 Private m_Y As Single

Private Sub Command1_Click()
   ' перерисовываем картинку
   Call DrawShapes(Picture1, cboFillColor)
End Sub

Private Sub Form_Load()
   Dim i As Long
   ' Начальная установка параметров и элементов управления
   ' Формирование списков
   Call FillColorList(cboFillColor)
   Call FillColorList(cboBorderColor)
   Call FillStyleList(cboFillStyle)
   Option1(1).Value = True
   'Для режима FLOODFILLSURFACE цвет границы не нужен
   cboBorderColor.Enabled = False
   BorderLabel.Enabled = False
   '
   Set Me.Icon = Nothing
   Me.Show
   Call DrawShapes(Picture1, cboFillColor)
   cboFillColor.ListIndex = 3
   cboBorderColor.ListIndex = 0
   cboFillStyle.ListIndex = 0
End Sub

Private Sub Picture1_Click()
   Dim MyBorderColor As Long
   Dim MyBorderColor1 As Long
   Dim MyFillColor As Long
   Dim MyFillStyle As FillStyleConstants
   Dim MyFlags As FillModes
   ' установка цвета и стиля заполнения
   MyFillColor = CheckSysColor _
        (cboFillColor.ItemData(cboFillColor.ListIndex))
   MyFillStyle = cboFillStyle.ItemData(cboFillStyle.ListIndex)
   MyFlags = Option1(0).Value + 1 ' режим заливки
   ' Формирование параметра BorderColor
   If MyFlags = FLOODFILLBORDER Then  ' цвет границы контура
     MyBorderColor = cboBorderColor.ItemData(cboBorderColor.ListIndex)
```

```
   End If
   ' выполняем заливку
   Call Paint(Picture1, m_X, m_Y, MyFillColor, _
     MyFillStyle, MyBorderColor, MyFlags)
End Sub

Private Sub Picture1_MouseDown _
    (Button As Integer, Shift As Integer, X As Single, Y As    Single)
   ' Фиксируем координаты нажатия мыши
   m_X = X
   m_Y = Y
End Sub

Private Sub Option1_Click(Index As Integer)
   If Index = 0 Then
     ' Для режима FLOODFILLBORDER гарантированный контур    "без дырок"
     ' сформирован в этом примере только СИНЕГО цвета
      cboBorderColor.ListIndex = 1
     ' но можно попробовать и другие цвета (может повезет)
      cboBorderColor.Enabled = True
      BorderLabel.Enabled = True
   Else 'Для режима FLOODFILLSURFACE цвет границы не нужен
      cboBorderColor.Enabled = False
      BorderLabel.Enabled = False
   End If
End Sub
```

Листинг 396-3. Модуль AddProc.bas — вспомогательные процедуры для формы Flood.frm

```
Option Explicit
 Private Declare Function GetSysColor Lib "user32" _
   (ByVal nIndex As Long) As Long
Function CheckSysColor(ByVal Color As Long) As Long
   Const HighBit As Long = &H80000000
   ' Если установлен старший разряд, то берем системный цвет
   If Color And HighBit Then
      CheckSysColor = GetSysColor(Color And Not HighBit)
   Else
      CheckSysColor = Color
   End If
End Function

Sub DrawShapes(pct As PictureBox, cbo As ComboBox)
  ' Начальная картинка с пересечением разных линий
  ' на рисунке
   Dim i As Long
   Dim x1 As Long, y1 As Long
   Dim x2 As Long, y2 As Long
   Dim c As Long
   Dim MyRnd As Single
   ' рисуем случайные линии разного цвета
   Randomize Timer
   With pct
      .Cls
      For i = 1 To 20
         x1 = Rnd * .ScaleWidth
         y1 = Rnd * .ScaleHeight
         x2 = Rnd * .ScaleWidth
         y2 = Rnd * .ScaleHeight
         If i <= 15 Then  ' для первых 15 выбираем случайные цвета
           MyRnd = Rnd * 7
```

```
         Else  ' для последних — фиксированный синий
           MyRnd = 1  'Синий
         End If
         c = cbo.ItemData(MyRnd)  ' выбор цвета
         ' прямая или круг (чет/нечет)
         If i Mod 2 Then
            pct.Line (x1, y1)-(x2, y2), c, B
         Else
            pct.Circle (x1, y1), y2, c
         End If
      Next i
   End With
End Sub
Sub FillStyleList(ByVal cbo As ComboBox)
   ' заполнение списка типом фактуры
   With cbo
      .AddItem "Сплошная заливка"
      .ItemData(.NewIndex) = vbFSSolid
      .AddItem "Горизонтальные линии"
      .ItemData(.NewIndex) = vbHorizontalLine
      .AddItem "Вертикальные линии"
      .ItemData(.NewIndex) = vbVerticalLine
      .AddItem "Верхняя диагональ"
      .ItemData(.NewIndex) = vbUpwardDiagonal
      .AddItem "Нижняя диагональ"
      .ItemData(.NewIndex) = vbDownwardDiagonal
      .AddItem "Клетка"
      .ItemData(.NewIndex) = vbCross
      .AddItem "Косая клетка"
      .ItemData(.NewIndex) = vbDiagonalCross
   End With
End Sub
Sub FillColorList(ByVal cbo As ComboBox)
   ' заполнение списка названиями цветов
   With cbo
      .AddItem "Черный"
      .ItemData(.NewIndex) = vbBlack
      .AddItem "Синий"
      .ItemData(.NewIndex) = vbBlue
      .AddItem "Циан"
      .ItemData(.NewIndex) = vbCyan
      .AddItem "Зеленый"
      .ItemData(.NewIndex) = vbGreen
      .AddItem "Лиловый"
      .ItemData(.NewIndex) = vbMagenta
      .AddItem "Красный"
      .ItemData(.NewIndex) = vbRed
      .AddItem "Белый"
      .ItemData(.NewIndex) = vbWhite
      .AddItem "Желтый"
      .ItemData(.NewIndex) = vbYellow
   End With
End Sub
```

Андрей Колесов, Ольга Павлова

© Андрей Колесов, Ольга Павлова, 2001
Авторский вариант. Статья была опубликована с незначительной литературной правкой в журнале "КомпьютерПресс" 10/2001, компакт-диск.

Совет 397. Как использовать свойство Filter при работе с ADO

При работе с наборами данных ADO вы можете использовать свойство Filter, например следующим образом:

```
rst.Filter = "pub_id ='000132' "
```

В этом случае будут выбраны все записи со значением 000132 в поле pub_id. Здесь важно помнить, что ADO физически не исключает из набора записи, которые не отвечают данному критерию, — просто они становятся недоступными для работы. Это, в частности, означает, что если вам нужно применить новый фильтр для работы с исходным набором, то нет необходимости производить отмену применения предыдущего фильтра — в любом случае ADO автоматически выполняет фильтрацию "исходного" набора данных.

Если же вы хотите вернуть набор данных в "неотфильтрованное" состояние, то нужно выполнить выборку с "пустым" фильтром:

```
rst.Filter = adFilterNone
```

Совет 398. В который раз повторяем: используйте режим Option Explicit

Мы уже неоднократно говорили о необходимости использовать режим обязательного объявления переменных, но приходится опять это повторять...

Читатель Вячеслав прислал письмо с вопросом, почему у него не работает вставка таблицы из VB-проекта в Word, точнее — почему не работает конструкция:

```
.ActiveDocument.Tables.Add
Selection.Range, NumRows:=2, NumColumns:=2
```

При выполнении этой конструкции выдается сообщение об ошибке Object required.

Судя по переписке (мы обменялись несколькими посланиями), на решение проблемы Вячеславу понадобилось больше недели, при этом он был уверен, что ее причина кроется в какой-то несовместимости VB и VBA. При этом он подозревал, что неадекватно ведет себя объект Range. Однако в данном случае все было гораздо проще: после переноса кода из VBA в VB Вячеслав забыл поставить точку перед Selection (этот код работал внутри конструкции With MyOblect).

Действительно, заметить такой дефект порой бывает непросто, а сообщение о необходимости объекта плохо помогает локализовать ошибку. Но если бы в программе был установлен режим Option Explicit, то появилось бы другое, более точное сообщение: "Не определена переменная Selection". К тому же ошибка была бы обнаружена не на этапе выполнения программы, а при компиляции кода. Очевидно, при такой диагностике понять причину неработоспособности кода было бы гораздо проще.

Напомним, что оператор Option Explicit будет автоматически вставляться в код нового программного модуля, если установить флажок Require Variable Declaration в окне Tools|Options|Editor (VB и VBA).

Совет 399. Задавайте вопросы в понятной форме

Именно с этого совета начиналась первая статья из серии "Размышления бывшего программиста" (КомпьютерПресс N 9/2000), но опять же приходится возвращаться к этой теме. Почему вопрос читателя, рассмотренный в предыдущем совете, решался больше недели? А потому, что его первое письмо содержало такой текст: "Я не знаю, как средствами не VBA, а VB6 создать таблицу в MS Word. Мне казалось, что все просто: скопировать содержимое макроса в процедуру. Однако VB6 ругается на именованный параметр Range. Подскажите, как мне быть?"

Кто сможет решить проблему, изложенную таким образом? Мы попросили прислать VB-проект с примером ситуации — Вячеслав прислал одну строку кода. Еще раз попросили... Только на третий раз он прислал нужный VB-проект, содержащий 7 строк кода (включая комментарии) и занимающий 1,5 Кбайт в ZIP-файле. Запустить пример, увидеть ошибку, понять причину и отправить ответ — все это заняло две минуты.

Итак, как нужно задать технический вопрос, если вы хотите действительно получить на него конкретный ответ:

- Отправляйте в приложении программный проект, который эксперт может реально запустить на своем компьютере. Если речь идет о VBA, то пришлите сам документ с вложенными в него макросами. Если файлов проекта несколько или они большого объема, лучше выслать в виде архивного ZIP-файла.
- Присылайте именно тестовый программный пример (а не проект огромного приложения), который содержит только код, позволяющий локализовать ошибку. Ничего лишнего!
- И почитайте дополнительные рекомендации, которые можно найти по адресу www.visual.2000.ru/develop/talks/talks1_1.htm.

Совет 400. Используйте режим раннего связывания

Об этой проблеме мы говорили в статье "Особенности технологий раннего и позднего связывания в Visual Basic"[85] (КомпьютерПресс N 9/2000). Оба режима имеют свои достоинства и недостатки, но общая рекомендация такова: если нет особой нужды применять позднее связывание (иногда это просто необходимо), то лучше использовать раннее связывание.

Вот какой код прислал нам Вячеслав:

```
 Dim wdApp As Object
   Set wdApp = CreateObject("word.application") 'открыть Word
     With wdApp
     .Documents.Add 'Создать новый документ
        .ActiveDocument.Tables.Add _
   Selection.Range, NumRows:=2, NumColumns:=2
End With
```

Здесь используется позднее связывание — конкретизация объекта wpApp происходит только в момент выполнения программы (CreateObject). Но предпочтительнее выглядит вариант с ранним связыванием:

```
Dim wdApp As Word.Application
```

[85] http://www.visual.2000.ru/develop/ms-vb/cp0009-1/binding.htm

```
 Set wdApp = New Word.Application
 With wrdApp
  .Documents.Add    ' новый документ
  .ActiveDocument.SaveAs App.Path &  _
     "\RTFDOC.doc",wdFormatDocument
End With
```

В этом случае мы сразу четко фиксируем тип объекта (нужно также установить ссылку на библиотеку Word 9.0 Object Library). И в такой ситуации можно воспользоваться всеми преимуществами интеллектуальных подсказок при вводе кода.

Совет 401. Управляйте режимами программного контроля ошибок

Традиционный вариант управления программным контролем ошибок выглядит примерно так:

```
Sub MyProcedure
  'установка программной обработки ошибок
 On Error {GoTo MyError|Resume Next}
 ...
  'отмена программной обработки
 On Error GoTo 0
 ...
End Sub
```

При выходе из процедуры (End Sub или Exit Sub) отмена программной обработки ошибок выполняется автоматически, поэтому многие программисты вообще не ставят оператор On Error GoTo 0. Но тут есть один подводный камень: если вы напишете такой код:

```
Sub MyProcedure
  'установка программной обработки ошибок
 On Error GoTo MyError
 Call OtherProcedure
 ...
MyError:
 ' обработка
  ...
End Sub
```

то в вызываемой подпрограмме OtherProcedure (и в других вложенных процедурах) будет продолжать действовать установка On Error GoTo MyError, то есть при появлении там ошибки управление будет передаваться на метку MyError. При этом нужно помнить, что если в процедуре OtherProcedure переопределить обработку ошибок, то при возврате управления в MyProcedure действие On Error GoTo MyError будет восстановлено автоматически.

Обычно мы применяем на практике децентрализованную обработку ошибок, когда в каждой "критической" подпрограмме реализуется свой локальный механизм обработки. Однако порой бывает полезным применять централизованную обработку, например в одном месте для всего приложения. Это легко сделать, имея в виду приведенный выше код:

```
Sub Main
  ' глобальное определение места
обработки ошибок
  ' для всего приложения
  On Error GoTo GlobalError
```

```
  Call OtherProcedure
 ...
GlobalError:
 ' обработка
  ...
End Sub
```

В какой бы из вложенных процедур ни произошла ошибка, управление будет передаваться в конструкцию GlobalError. Но в этом случае возникает другая проблема: как определить, где конкретно произошла ошибка и откуда передано управление? Это можно легко решить, создав глобальную переменную:

```
Public MyErrorPoint As String
 Sub MyProcedure
  On Error GoTo GlobalError
  ...
  MyErrorPoint = "MySub"
  Call MySub
GlobalError:
 Select Case MyErrorPoint
   ...
```

И еще одна рекомендация: при прочих равных условиях предпочтительнее выглядит включение обработки ошибок с автоматической передачей управления на следующий оператор. Тогда вы сами можете проверять наличие ошибок в критических точках программы:

```
 On Error Resume Next
 ...
 Open "TestFile" For Input As #1
 If Err.Number <> 0 Then  ' ошибка
```

Совет 402. Быстрый поиск по списку

При выборе позиции списка может пригодиться вариант ввода с клавиатуры первых символов искомой строки. Это можно реализовать с помощью такого кода:

```
Private Declare Function SendMessage _
   Lib "user32" Alias "SendMessageA" _
   (ByVal hwnd As Long, ByVal wMsg As Long, _
   ByVal wParam As Long, lParam As Any) As Long

Private sSearch As String
Private Const LB_FINDSTRING = &H18F
Private Const lb_Err = (-1)

Private Sub Form_Load()
  ' заполнение списка
  With List1
    .AddItem "Adam"
    .AddItem "Allan"
    .AddItem "Arty"
    .AddItem "Aslan"
    .AddItem "Barbey"
    .AddItem "Bob"
  End With
  Timer1.Interval = 2000  ' сброс каждые две секунды
End Sub
```

```
Private Sub List1_KeyPress(KeyAscii As Integer)
   ' обработка нажатых клавиш
   Dim nResult As Long
   Timer1.Enabled = True   ' запуск контроля по таймеру
   sSearch = sSearch + Chr$(KeyAscii)
   With List1
     ' поиск по списку
     nResult = SendMessage(.hwnd, LB_FINDSTRING, _
              .ListIndex, ByVal sSearch)
     If nResult <> lb_Err Then ' нашли
       .ListIndex = nResult ' установка позиции
       KeyAscii = 0  ' сброс ввода клавиши
     End If
   End With
End Sub

Private Sub Timer1_Timer()
   ' сброс поиска каждые две секунды
   sSearch = ""
   Timer1.Enabled = False
End Sub
```

Здесь реализован вариант, когда на ввод отводится только две секунды. Но можно придумать и другой механизм инициализации строки поиска.

Совет 403. При программном создании наборов данных указывайте размеры полей

Если вы попробуете выполнить такой код при программном создании набора данных ADO:

```
Dim rsBuild As ADODB.Recordset
Set rsBuild = New ADODB.Recordset
rsBuild.CursorLocation = adUseClient
rsBuild.Fields.Append "Question", adLongVarWChar
rsBuild.Fields.Append "AnswerA", adLongVarWChar
```

то, скорее всего, получите сообщение об ошибке: "Run-Time Error 3001: Method 'Append' of object 'Fields' failed". В документации VB по поводу метода Append говорится, что размер поля является необязательным параметром. Но это неверно — его нужно указывать в явном виде. Поэтому приведенный далее код будет работать без проблем:

```
Private Sub Form_Load()
  Dim rsBuild As ADODB.Recordset
  Set rsBuild = New ADODB.Recordset
  With rsBuild
    .CursorLocation = adUseClient
    .Fields.Append "Question", adLongVarWChar, 1
    .Fields.Append "Answer", adLongVarWChar, 1
    .Open
    .AddNew 0, String(5000, "*")
    .Update
    Debug.Print .Fields(0)
    .Close
  End With
  Set rsBuild = Nothing
End Sub
```

Совет 404. Программное отключение предупреждающих сообщений в приложениях MS Office

При использовании механизма Office Automation в VB-приложениях часто бывает желательно отключать выдачу предупреждающих сообщений. Например, когда вы программно удаляете рабочий лист Excel, то, возможно, не захотите получать запрос о том, нужно ли действительно выполнить эту операцию.

Пользователи Microsoft Access знают, что такое отключение в данном приложении можно выполнить следующим образом:

```
DoCmd.SetWarnings = False
```

Этот код выключает выдачу внутренних программных сообщений. Однако объект DoCmd — это возврат к временам, когда офисные приложения имели разные внутренние языки программирования. Такого объекта в других приложениях Office нет, но есть свойство DisplayAlerts, которое решает именно эту задачу. Обратите внимание, что его поддержка выполняется в различных программах по-разному. Например, в Excel этому свойству нужно просто присвоить значение логической переменной:

```
Public Sub DeleteWorksheet()
    ' запретить выдачу  сообщений
  Application.DisplayAlerts = False
  ActiveWindow.SelectedSheets.Delete
  Application.DisplayAlerts = True  'разрешить
End Sub
```

Следует иметь в виду, что нужно восстановить режим выдачи подобных сообщений, так как Excel не делает этого автоматически после завершения макрокоманды. Word для установки свойства DisplayAlerts использует специальные встроенные константы: wdAlertsNone, wdAlertsAll and wdAlertsMessageBox. Смысл первых двух установок понятен из их названий, а последняя означает, что Word будет выдавать только стандартные сообщения. Выполнение в этом случае кода:

```
ActiveDocument.Close
```

вызовет появление вопроса о сохранении документа. При выполнении следующего кода:

```
Application.DisplayAlerts = wdAlertsNone
ActiveDocument.Close
Application.DisplayAlerts = wdAlertsAll
```

сразу же появится окно SaveAs без промежуточного вопроса.

Совет 405. Как выводить на экран формы в динамическом режиме

Допустим, вы желаете выводить на экран те или иные формы, имена которых записаны в каком-то списке. Для этого можно воспользоваться коллекцией Forms, выбор из которой нужной формы выполняется с помощью индекса или имени:

```
Set frm = Forms(1)
```

или:

```
Set frm = Forms("frmEditor")
```

Соответственно вывод нужной формы по имени из списка может выглядеть следующим образом:

```
Private Sub Form_Load()

  With List1
    .AddItem "Form1"
    .AddItem "Form2"
    .AddItem "Form3"
   End With
End Sub

Private Sub List1_Click()
  Dim frm As Form
  Dim selForm As String
  selForm = List1.List(.ListIndex)
  Set frm = Forms.Add(selForm)
  frm.Show
  Set frm = Nothing
End Sub
```

Совет 406. Как узнать параметры диска

Вы хотите узнать серийный номер жесткого диска, а заодно и имя тома логического диска? Это можно сделать с помощью такого кода:

```
Private Declare Function GetVolumeInformation _
  Lib "kernel32" Alias "GetVolumeInformationA" _
  (ByVal lpRootPathName As String, _
  ByVal lpVolumeNameBuffer As String, _
  ByVal nVolumeNameSize As Long, _
  lpVolumeSerialNumber As Long, _
  lpMaximumComponentLength As Long, _
  lpFileSystemFlags As Long, _
  ByVal lpFileSystemNameBuffer As String, _
  ByVal nFileSystemNameSize As Long) As Long

Private Sub Form_Load()
   Dim sDrive As String
   Dim VolumeName As String
   Dim SerialNumber As Long
   '
   sDrive = "c:\"  'имя диска
   VolumeName = Space$(128)
   Call GetVolumeInformation(sDrive, _
      vbNullString, 128&, SerialNumber, _
      ByVal 0&, ByVal 0&, vbNullString, 0)
   VolumeName = Left$(VolumeName, InStr(VolumeName, Chr$(0)))
   MsgBox "Серийный номер диска = " & SerialNumber & vbCrLf & _
          "Имя тома = " & VolumeName
End Sub
```

Совет 407. Как создать дубликатный набор данных

Следует иметь в виду, что метод Clone не создает дубликат набора данных — он просто делает два разных указателя, которые работают с одним и тем же физическим набором. А

выполнить формирование копии можно с помощью объекта Stream из состава ADO 2.5 (или более поздней версии):

```
Dim rsOne As New ADODB.Recordset
Dim rsTwo As New ADODB.Recordset
Dim oTempStream As New ADODB.Stream
' Далее идет формирование набора reOne
 ...
rsOne.Save
oTempStream, adPersistXML  ' запоминаем
rsTwo.Open
oTempStream            ' восстанавливаем
```

Совет 408. Как узнать размер свободного места на диске

Для этого можно использовать API-функцию GetDiskFreeSpaceA из библиотеки Kernel32, но она корректно работает для дисков размером до 2 Гбайт, так как была предназначена для Windows 95, где использовали FAT16. При работе с дисками больших размеров можно применить объект FileSystemObject из состава библиотеки Microsoft Scripting Runtime library (scrrun.dll), которую можно загрузить по адресу: http://www.microsoft.com/msdownload/vbscript/scripting.asp. Вот как будет выглядеть нужный нам код:

```
Dim fso As FileSystemObject
Dim drv As Drive
Set fso = New FileSystemObject
Set drv = fso.GetDrive("C")
MsgBox FormatNumber(drv.AvailableSpace / 1024, 0) & " Кб"
```

Совет 409. Блокировка Windows 2000

Осуществление блокировки Windows NT оказалось делом непростым. Но Windows 2000 включает в себя специальную функцию для выполнения такой операции:

```
Private Declare Function LockWorkStation Lib _
   "user32.dll" () As Long
Call LockWorkStation
```

Поскольку у этой функции нет параметров, то к ней можно обратиться напрямую следующим образом:

```
Call Shell ("rundll32 user32.dll, LockWorkStation", vbNormalFocus)
```

Такую конструкцию можно применять и в 16-разрядных приложениях. А вот как будет выглядеть операция на VBScript:

```
Dim WshShell
Set WshShell = CreateObject("WScript.Shell")
WshShell.Runl ("rundll32 user32.dll, LockWorkStation)
```

Андрей Колесов, Ольга Павлова

© Андрей Колесов, Ольга Павлова, 2002
Авторский вариант. Статья была опубликована с незначительной литературной правкой в журнале "КомпьютерПресс" 01/2002, компакт-диск.

Совет 410. Настройка интерфейса VB/VBA

Далеко не все VB/VBA-разработчики знают о возможности выполнять настройку интерфейса среды разработки с помощью окна Customize, которое открывается командой View|Toolbars|Customize (рис. 410-1).

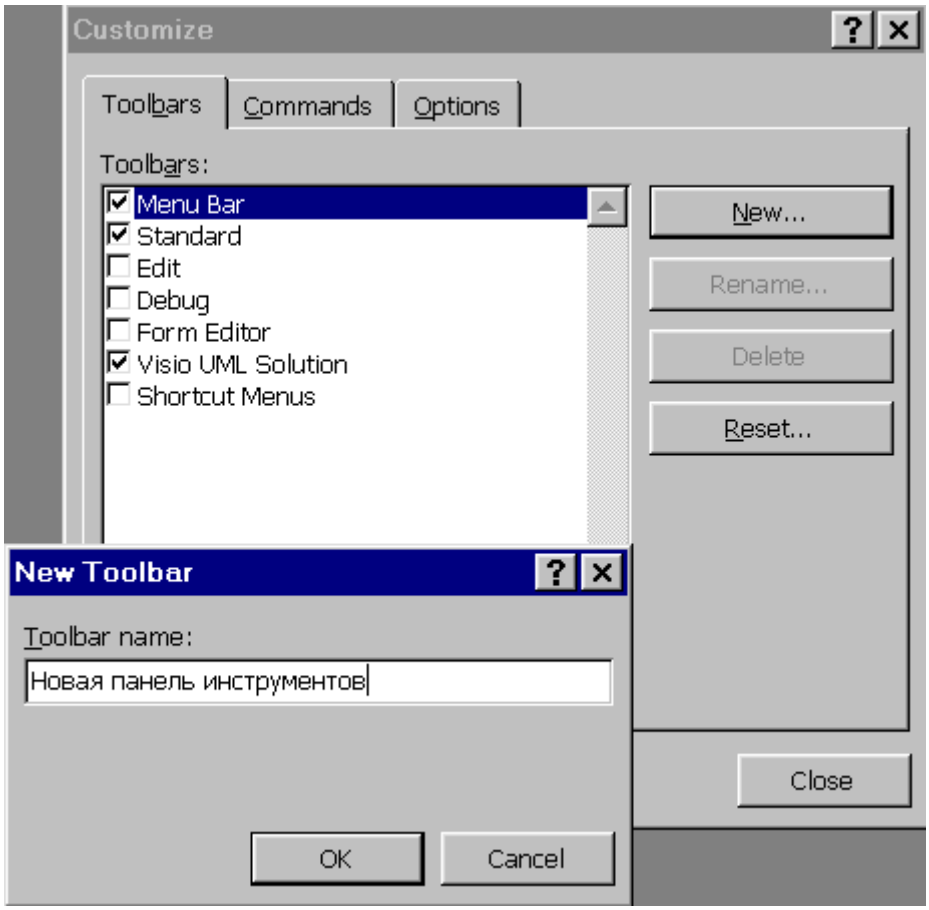


Рис. 410-1

Как и в офисных приложениях, вы можете делать видимыми или невидимыми панели инструментов (многие программисты думают, что существует только одна — Стандартная), создавать собственные панели, перемещать команды между разными панелями и меню, то есть создавать такую систему меню и панелей инструментов, какая вам кажется наиболее подходящей.

Советуем также открыть вкладку Customize|Commands (рис. 410-2). В списке Categories вы обнаружите перечень категорий команд, которые представлены в среде в виде меню. Но если внимательно посмотреть на состав команд отдельных категорий (список Commands),

то можно заметить, что на самом деле их гораздо больше, чем видны в меню по умолчанию.

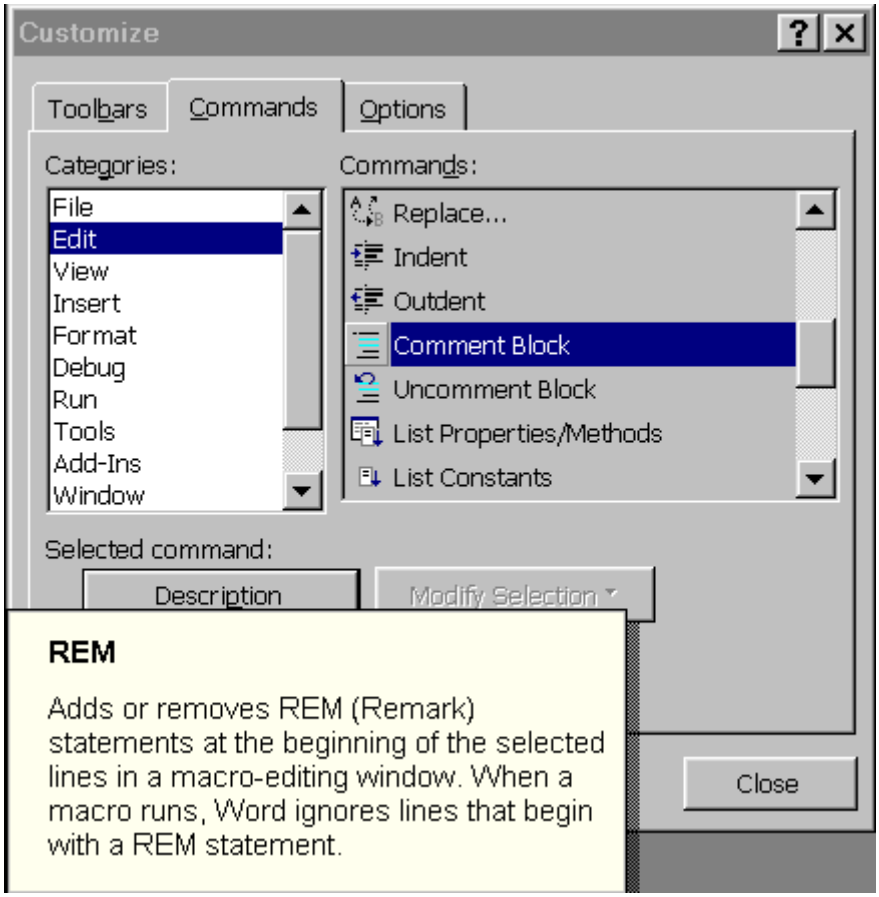


Рис. 410-2

К сожалению, справка окна Customize работает не самым лучшим образом, поэтому разобраться в смысле незнакомых команд непросто. Пользователи офисных приложений находятся в лучшем положении: в справке имеется раздел VB Use Interface Help с описаниями всех встроенных команд. Они также могут воспользоваться кнопкой Description для вывода подсказки (в VB эта кнопка почему-то не работает).

Совет 411. Добавьте команды управления комментариями к среде разработки

Продемонстрируем, как применить предыдущий совет[86] на практике. Так, при разработке приложений довольно часто бывает нужно закомментировать целые блоки программного кода. Это приходится делать в тех случаях, когда нужно на некоторое время удалить код или даже удалить совсем, но оставить его в виде комментария. Обычно вы это делаете, пробегая курсором по всем строкам и нажимая клавишу апострофа. Но все это можно гораздо проще совершить с помощью малоизвестных для вас команд среды разработки VB.

Командой View|Toolbars|Customize откройте окно Customize|Commands. Далее в списке Categories выделите строку Edit. Потом в списке Commands найдите команды Comment Block и Uncomment Block, а затем перетащите их мышью на панель управления Standard.

[86] http://www.visual.2000.ru/develop/ms-vb/tips/0201.htm

Теперь, если вы выделите некоторый фрагмент кода, то одним щелчком мыши по соответствующей команде сможете устанавливать или убирать апострофы в первой позиции выделенных строк.

Кстати, среди таких малоизвестных команд имеются также Indent и Outdent, которые позволяют сдвигать фрагменты текста на один отступ вправо или влево.

Совет 412. Как прочитать адреса отправленных писем

Допустим, вам нужно получить список адресов, по которым вы отправили письма. Если они лежат в папке отправленных писем Outlook, то это можно сделать с помощью такого кода:

```
Dim myItems As Items
Dim mailmsg As MailItem
' выборка писем из папки
Set myItems = _
   Application.Session.GetDefaultFolder(olFolderOutbox).Items

If myItems.Count > 0 Then ' есть письма
  For Each mailmsg In myItems
    MsgBox "Куда отправлено - " & mailmsg.To
  Next
Else
  MsgBox "нет отправленных писем"
End If
```

Если же вам нужно обратиться к некоторой произвольной пользовательской папке, то выборка писем будет выглядеть примерно так:

```
Set myItems = Application.GetNamespace("MAPI"). _
    Folders.Item("Personal Folders").Folders.Item("Outbox").Items
```

Совет 413. Использование пробелов в именах таблиц баз данных ADO

Наверняка вам известно, что в именах таблиц и полей баз данных ADO нельзя использовать ни пробелы, ни идентификаторы типа "Большая деревня". Но на самом деле подобные имена вполне допустимы, просто для их обозначения нужно применять квадратные скобки. Соответственно метод Open будет выглядеть примерно следующим образом:

```
rst.Open "[Большая деревня]", conn,,,adCmdTable
```

А SQL-запрос может выглядеть так:

```
rst.Open "SELECT [Малыш Дик] FROM [Большая деревня]"
```

Совет 414. Как передать Null в API-функцию

Довольно часто бывает необходимо передать в API-функцию значение Null (0&) в качестве одного из аргументов. Но сделать это порой не так-то просто, поскольку вы можете столкнуться с ограничением объявления функции: VB выдаст в этом случае сообщение об ошибке. Например, рассмотрим такое объявление функции, которое

получено копированием соответствующего кода из файла Win32api.txt с помощью API Viewer:

```
Public Declare Function ScrollWindowEx Lib _
  "user32" (ByVal hwnd As Long, ByVal dx As Long, _
  ByVal dy As Long, lprcScroll As RECT, _
  lprcClip As RECT, ByVal hrgnUpdate As Long, _
  lprcUpdate As RECT, ByVal fuScroll As Long) As Long
```

Здесь видно, что эта функция использует несколько структур RECT, в которых задается область прокрутки Windows. Но если передать вместо адресов блоков данных нуль, то Windows самостоятельно будет определять эти области. С этих позиций следующее обращение к функции является вполне допустимым:

```
ScrollWindowEx m_hWnd, 0, 5, ByVal 0&, ByVal 0&, 0&, ByVal 0&, _
  SW_SCROLLCHILDREN Or SW_INVALIDATE
```

Однако в данном случае VB выдаст сообщение о том, что такое обращение (передача аргумента по значению) противоречит сделанному выше объявлению (передача по ссылке).

Чтобы решить указанную проблему, можно использовать два варианта. В первом нужно заменить в объявлении типы данных RECT на Any:

```
Public Declare Function ScrollWindowEx Lib _
  "user32" (ByVal hwnd As Long, ByVal dx As Long, _
  ByVal dy As Long, lprcScroll As Any, _
  lprcClip As Any, ByVal hrgnUpdate As Long, _
  lprcUpdate As Any, ByVal fuScroll As Long) As Long
```

С таким объявлением будет допустима передача как адреса структуры, так и числовой величины.

Однако, как мы уже неоднократно упоминали, использование типа Any повышает риск ошибки (фактически в данном случае отключается синтаксический контроль, вследствие чего вся ответственность за правильность обращения возлагается на программиста). Поэтому мы советуем воспользоваться возможностью альтернативного объявления функции, например следующим образом:

```
Public Declare Function ScrollWindowExLong Lib _
  "user32" Alias ScrollWindowEx
  (ByVal hwnd As Long, ByVal dx As Long, _
  ByVal dy As Long, ByVal lprcScroll As Long, _
  ByVal lprcClip As Long, ByVal hrgnUpdate As Long, _
  ByVal lprcUpdate As Long, ByVal fuScroll As Long) As Long
```

Соответственно при необходимости передачи нулевых параметров нужно использовать обращение именно к этой функции:

```
ScrollWindowExLong _
  m_hWnd, 0, 5, ByVal 0&, ByVal 0&, 0&, ByVal 0&, _
  SW_SCROLLCHILDREN Or SW_INVALIDATE
```

Совет 415. Как вставить колонки в VB MSFlexGrid

Элемент управления MSFlexGrid предоставляет отличные возможности для визуализации данных в виде таблицы. Однако иногда необходимо, чтобы сам пользователь мог вставлять колонки в определенных точках таблицы, в то время как VB по умолчанию добавляет новые колонки в конец таблицы. Эта проблема легко решается с помощью обновленного свойства ColPosition, для чего достаточно написать такой код:

```
Private Sub Command1_Click()
  With MSFlexGrid1
    ' добавить новую колонку
    .Cols = .Cols + 1
    ' переместить в нужную позицию
    .ColPosition(.Cols - 1) = .Col
  End With
  Call SetHeaders  ' переписать заголовки
End Sub

Private Sub SetHeaders()
  ' формирование заголовков
  Dim i As Integer
  With MSFlexGrid1
    For i = 0 To .Cols - 1
      .Col = i
      .Row = 0
      .Text = "Колонка " & i
    Next i
  End With
End Function
```

Совет 416. Динамическое увеличение динамического массива

Если вам нужно динамически увеличить верхнюю границу массива, то можно воспользоваться такой конструкцией:

```
Dim upper As Long ...

upper = UBound(myArray)    ' текущее значение верхней границы
ReDim myArray(upper + 1)  'увеличение
```

Но если к моменту обращения к функции Ubound массив не был еще инициализирован (то есть был определен в виде Dim myArray()), то приведенный выше код вызовет ошибку. Чтобы избежать этого, используйте такой код:

```
Dim myArray() As String

Private Sub cmdAdd_Click()
  Dim upper As Long

  On Error Resume Next  ' включаем обработку ошибок
  upper = UBound(myArray)
  If Err.Number Then
    upper = 0
    ReDim myArray(0)
  Else
    upper = upper + 1
    ReDim Preserve myArray(upper)
  End If
  On Error GoTo 0   ' отключаем обработку ошибок
```

```
   myArray(upper) = CStr(txtName) ' ввод нового значения
End Sub
```

Совет 417. Как работать с адресной книгой MS Exchange

Наш читатель Виктор Крюков поделился своим опытом работы с адресной книгой MS Exchange. В организации, где он работает, почтой управляет MS Exchange 2000, поэтому результат обращения к свойству ContactItem.Email1Address выглядит примерно так:

```
"/o=UFG/ou=UFG/cn=Recipients/cn=VKryukov".
```

Как же получить реальный адрес? Виктор предлагает такой рецепт.

Объект, который хранит информацию о пользователе, — это AddressEntry, находящийся в коллекции AddressEntries, которая, в свою очередь, находится в объекте AddressList коллекции AddressLists данной сессии. У этого объекта есть семейство Fields, которое и хранит нужные свойства. Видимо, для доступа эти свойства проиндексированы какими-нибудь именованными константами типа PR_ADDRESS_EMAIL и т.п., но я их не нашел.

Вот возможные варианты решения этой задачи:

1. *делать цикл по всем полям и искать значение, содержащее '@' в качестве подстроки;*
2. *найти значение, которое является массивом, — оно всего одно, представляет собой массив строк и содержит строчки примерно такого вида (этот вариант кажется более надежным):*

```
3.    SMTP:_servrice@ufg.com
4.    MS:UFG/UFG/Service
5.    CCMAIL: Service at UFG
6.    X400:c=RU;a= ;p=UFG;o=UFG;s=?service;
```

   то есть это адреса в разных форматах. Из данного массива выбирается строчка, начинающаяся с 'SMTP', и обрезается начало, которое и является правильным ответом.

Вот код для примера (я добавляю в коллекцию Result все имена и адреса), для выполнения которого требуется в References подключить Microsoft CDO 1.21 Library.

```
Private Sub Form_Load()
  Dim oSession As MAPI.Session
  Dim oAList As MAPI.AddressList
  Dim oAEntry As MAPI.AddressEntry
  Dim oValue As Variant
  Dim i As Integer, v As Variant, j As Integer
  Dim Result As New Collection

  ' Logon to the MAPI session
  Set oSession = New MAPI.Session
  oSession.Logon

  ' Get the Global Address List
  Set oFolder = oSession.AddressLists("Global Address List")
  For Each oEntry In oFolder.AddressEntries
    For i = 1 To oEntry.Fields.Count
      v = oEntry.Fields(i).Value
      If IsArray(v) Then
```

```
        For j = LBound(v) To UBound(v)
         If Mid(v(j), 1, 5) = "SMTP:" Then
           Result.Add Mid(v(j), 6) & " : " & oEntry.Name
           GoTo continue
         End If
        Next j
      End If
    Next i
continue:
  Next oEntry

  For i = 1 To Result.Count
    MsgBox Result(i)
  Next i
End Sub
```

Совет 418. Как отключить ADO Recordset, созданный с помощью объекта Command

Когда вы создаете распределенное приложение, крайне важно, чтобы ваш код занимал как можно меньше ресурсов сервера. В этом случае узким местом является поддержка соединения с сервером баз данных. Конечно, выполнить отключение набора данных из стандартного объекта Recordset очень просто. А вот сделать это с набором данных, который был сгенерирован с помощью объекта Command, не так легко, хотя и возможно.

В этих целях создайте объект Command и установите его свойства, как вы это обычно делаете, но вместо обращения к свойству Execute используйте метод Open объекта Recordset с аргументом Source. Тогда вы отключите соединение обычным образом и удалите объект Command. Все это можно проиллюстрировать с помощью такого кода:

```
 Dim cmd As ADODB.Command
 Dim rst As ADODB.Recordset
 Dim param As ADODB.Parameter

Set cmd = CreateObject("ADODB.Command")
  With cmd
    .CommandText = "sp_myStoredProc"
    .CommandType = adCmdStoredProc
    .ActiveConnection = STR_CONN  'Какая-то строка соединения
    Set param = .CreateParameter("Param1", _
                adVarChar, adParamInput, 2, "foo")
    .Parameters.Append param
  End With
  Set rst = CreateObject("ADODB.Recordset")
  With rst
    .CursorLocation = adUseClient
    .Open cmd, CursorType:=adOpenStatic, _
            Options:=adCmdStoredProc
    Set .ActiveConnection = Nothing
  End With
  Set DataGrid1.DataSource = rst
  Set cmd = Nothing
End Sub
```

Совет 419. Как использовать символ @ в именах параметров SQL Server

Как известно, ADO предлагает два варианта поддержки хранимых процедур со значениями параметров: в первом случае нужно сослаться на имя параметра напрямую и

просто присвоить значение, а во втором следует построить объект Parameter и добавить его к коллекции Parameters объекта Command. Однако нужно иметь в виду, что, когда вы используете именованные параметры, следует применять символ @ перед именем каждого параметра SQL, например вот так:

```
cmd.Parameters("@someParam") = 250
```

Если же вы создаете объект Parameter, этот символ не используется:

```
Set param = cmd.CreateParameter _
  ("someParam", adBigInt, adParamInput, 2, 250)
cmd.Parameters.Append param
```

Совет 420. Генерация строк соединения для OLE DB

Создать строку соединения OLE DB без использования элемента управления DataEnvironment в VB-проекте не так-то просто: нужно в контекстном меню объекта Connection1 выделить команду Properies, а затем установить значение в диалоговом окне Data Link Properties, которое потом переписать в свойство ConnectionSource. Однако существует довольно легкое решение этой проблемы.

Создайте текстовый файл с расширением VBS (VBScript) с таким кодом:

```
Dim oDataLinks, sRetVal
 Set oDataLinks = CreateObject("DataLinks")
 On Error Resume Next  'отслеживание кнопки Cancel
 sRetVal = oDataLinks.PromptNew
 On Error Goto 0  ' отключение обработки ошибок
 If Not IsEmpty (sRetVal) Then  ' не была нажата Cancel
   InputBox "Строка соединения OLE DB = ", sRetVal
 End If
 Set oDataLinks = Nothing
```

После двойного щелчка по имени этого файла начнет выполняться этот код, и вы сможете скопировать строку соединения из поля ввода.

Если же вы работаете с VB, то можно добавить ссылку на Microsoft OLE DB Service Component 1.0 Type Library (OLEDB32.DLL) и использовать Object Browser для обращения к объекту DataLinks.

Совет 421. Как создавать ISAM-файлы

С помощью библиотеки ADOX 2.1 (Microsoft ADO Extensions for DLL and Security) можно легко создать новый ISAM-файл в самых различных форматах (Excel, dBase, Paradox, HTML и Lotus) без обращения к соответствующим объектным моделям и даже при отсутствии приложений, для которых эти форматы являются родными.

Приведем пример того, как можно создать XLS-файл с двумя рабочими листами:

```
Public Sub Main()
  Dim cn As Connection
  Dim cat As Catalog
  Dim tbl As Table
  Dim fld As Column
```

```
  Set cn = New Connection
  With cn  ' установка соединения
    .ConnectionString = _
      "Provider=Microsoft.Jet.OLEDB.4.0;" & _
      "Extended Properties=Excel 8.0;" & _
      "Data Source=" & App.Path & "\newfile.xls"
    .Open
  End With

  Set cat = New Catalog
  With cat  ' формирование таблиц
    .ActiveConnection = cn

    ' первая таблица (рабочий лист)
    Set tbl = New Table
    tbl.Name = "NewSheet1"

    Set fld = New Column
    fld.Name = "MyCol1"
    fld.Type = adWChar
    fld.DefinedSize = 30
    tbl.Columns.Append fld
    ' вторая колонка
    tbl.Columns.Append "Column2", adInteger

    .Tables.Append tbl  ' добавляем первую таблицу

    ' вторая таблица (рабочий лист)
    Set tbl = New Table
    tbl.Name = "NewSheet2"

    ' вторая колонка
    tbl.Columns.Append "МояКолонка1", adWChar, 20

    .Tables.Append tbl  ' добавляем вторую таблицу

    cat.Tables.Refresh  ' запись
  End With
  ' очистка
  Set fld = Nothing
  Set tbl = Nothing
  Set cat = Nothing
  cn.Close
  Set cn = Nothing
End Sub
```

Запустите эту процедуры, а потом с помощью Excel убедитесь, что вы получили рабочую книгу с двумя листами, в которых заданы соответственно две и одна колонки. Вы можете создавать файлы и других типов, просто установив в Extended Properties нужное значение: dBase IV, Paradox 4.x и пр.

Совет 422. Защита от макросов при использовании глобальных шаблонов

К нам поступил вопрос: *"Почему при работе в Word 2000 при использовании глобальных шаблонов, а также при загрузке шаблонов в виде автономных документов или присоединенного файла из каталога "Шаблоны пользователя" не производится проверка на наличие макрокода, хотя такой режим контроля у меня установлен?"*

Отвечаем на это следующим образом.

Управление режимом контроля за наличием макрокода при загрузке документов производится в диалогом окне Security[Безопасность] (команда меню Tools|Macro|Securiry [Сервис/Макрос/Безопасность]). На его вкладке Trusted Sources [Надежные источники] имеется флажок Trust all installed Add-ins and templates [Доверять всем установленным надстройкам и шаблонам]. Под установленными здесь подразумеваются дополнения и шаблоны, помещенные в каталог "Шаблоны пользователя" (конкретное имя этого каталога указывается в поле User Templates [Шаблоны пользователя] во вкладке File Locations [Расположение] диалогового окна Tools|Options [Сервис|Параметры]).

По умолчанию данный флажок установлен, поэтому все шаблоны из этого каталога не проверяются на наличие макрокода (предполагается, что вы помещаете туда файлы, в которых точно уверены). Если вы все же хотите выполнять такую проверку, то очистите флажок.

В Word 97 специального режима для загрузки шаблонов не было. Но в начальной версии программы иногда имела место ошибка, когда шаблоны загружались без проверки на макрокод. Этот дефект уже давно устранен — заплатку, которая решает данную проблему, можно скачать по адресу www.microsoft.com/rus/download/wd97sp.htm[87].

Андрей Колесов, Ольга Павлова

© Андрей Колесов, Ольга Павлова, 2002
Авторский вариант. Статья была опубликована с незначительной литературной правкой в журнале "КомпьютерПресс" 03/2002, компакт-диск.

Совет 423. Управление подключением макросов в приложениях Office

В одной из Web-дискуссий был задан такой вопрос: "При загрузке Outlook у меня постоянно выдается окно предупреждения о наличии макросов в загружаемом проекте. Как мне добиться того, чтобы загрузка выполнялась автоматически без требования подтверждения в диалоговом окне?"

В целом решение этого вопроса одинаково для всех приложений Office. Но в отношении Outlook (версии 2000 и 2002) стоит напомнить, что эта программа использует для хранения макросов только один фиксированный файл с именем VbaProject.OTM, который хранится в каталоге C:\Windows\Application Data\Microsoft\Outlook\. Сам программный проект может иметь еще и собственное произвольное имя (оно видно только в среде VBA).

Для ответа на поставленный вопрос нужно иметь в виду, что приложения Office (начиная с 2000) имеют три уровня безопасности для управления загрузкой макросов, а также возможность использования цифровой подписи. С помощью этих механизмов можно более гибко управлять режимами загрузки.

Рассмотрим возможности управления безопасностью. Режим защиты устанавливается в диалоговом окне "Безопасность" (Security), которое открывается командой

[87] http://www.microsoft.com/rus/download/wd97sp.htm

"Сервис|Макро|Безопасность" (рис. 423-1), где видно описание трех возможных уровней безопасности.

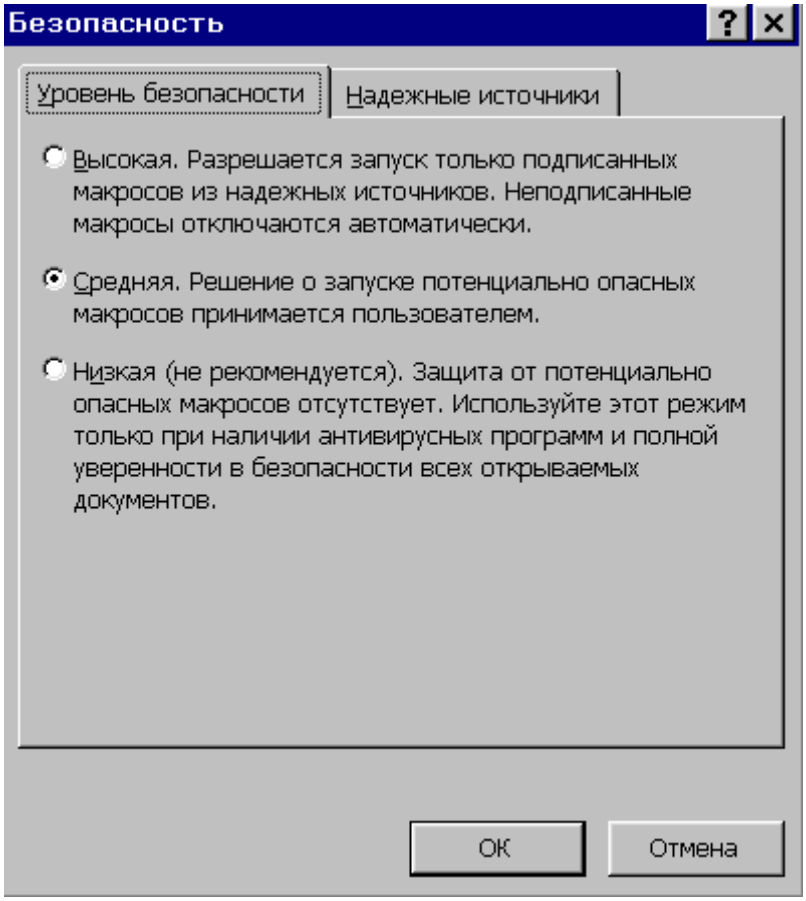


Рис. 423-1

1. **Высокий.** Разрешается запуск только подписанных макросов из надежных источников. Неподписанные макросы удаляются автоматически. При наличии макросов с неизвестными подписями выдается окно предупреждения, однако такие макросы можно подключить, только признав подлинность подписи.
2. **Средний.** Подписанные макросы загружаются автоматически. О наличии неподписанных макросов выдается предупреждение, и решение об их загрузке принимается пользователем.
3. **Низкий.** Защита отсутствует, все макросы загружаются автоматически.

В приложениях Office по умолчанию установлен средний уровень безопасности, что и вызывает появление окна предупреждения о наличии макросов. Соответственно существует два варианта ответа на приведенный выше вопрос:

1. Установите низкий уровень безопасности — и никаких предупреждений вообще не будет. Но мы не рекомендуем этот вариант, так как проверка на наличие макросов при загрузке неизвестных файлов (например, полученных из Интернета или по электронной почте) необходима. Но для Outlook такой вариант является вполне подходящим — ведь мы имеем дело только с фиксированным файлом локального компьютера, который создается исключительно его хозяином.

2. Лучший способ (наиболее универсальный) — использовать цифровую подпись проекта (в том числе и для Outlook). Этот способ немного подробнее мы рассмотрим в совете 424[88].

Для тех, кто работает с Word и Excel, есть еще один вариант, который позволяет отменить проверку наличия макросов в глобальных шаблонах и Add-ins. Для подобных приложений во вкладке Trusted Sources [Надежные источники] окна Security [Безопасность] имеется флажок Trust all installed Add-ins and templates [Доверять всем установленным надстройкам и шаблонам].

Под "установленными" подразумеваются дополнения и шаблоны, помещенные в каталог "Шаблоны пользователя". (Конкретное имя этого каталога указывается в поле User Templates [Шаблоны пользователя] во вкладке File Locations [Расположение] диалогового окна Tools|Options [Сервис|Параметры].)

По умолчанию данный флажок установлен, поэтому все шаблоны из этого каталога не проверяются на наличие макрокода (подразумевается, что вы помещаете туда файлы, в которых абсолютно уверены). Если вы все же хотите выполнять такую проверку, то снимите флажок.

В Word 97 такого специального режима для загрузки шаблонов не было. Но в начальной версии программы иногда имела место ошибка, когда шаблоны загружались без проверки на макрокод. Этот дефект уже давно устранен — заплатку, которая решает данную проблему, можно скачать по адресу www.microsoft.com/rus/download/wd97sp.htm[89].

Совет 424. Использование цифровой подписи в Office 2000/XP

Здесь есть три элемента: создание и удаление сертификата, подключение цифровой подписи к документу, признание подписи в качестве "надежной". Рассмотрим их последовательно.

1. **Создание и удаление сертификата.**

   Каждый пользователь может создать один или несколько сертификатов. Это делается тремя способами: создать собственный сертификат, получить сертификат у администратора сети (если на предприятии разработаны собственные стандарты) и получить фирменный сертификат в специализированном центре (например, у той же компании Microsoft). Собственный сертификат создается утилитой SelfCert.exe, которая входит в состав пакета (рис. 424-1). В первом варианте русской версии MS Office 2000 в этой программе была ошибка (не раскрывалось диалоговое окно), но уже в первом сервисном наборе обновления дефект был исправлен.

[88] http://www.visual.2000.ru/develop/ms-vb/tips/0203.htm
[89] http://www.microsoft.com/rus/download/wd97sp.htm

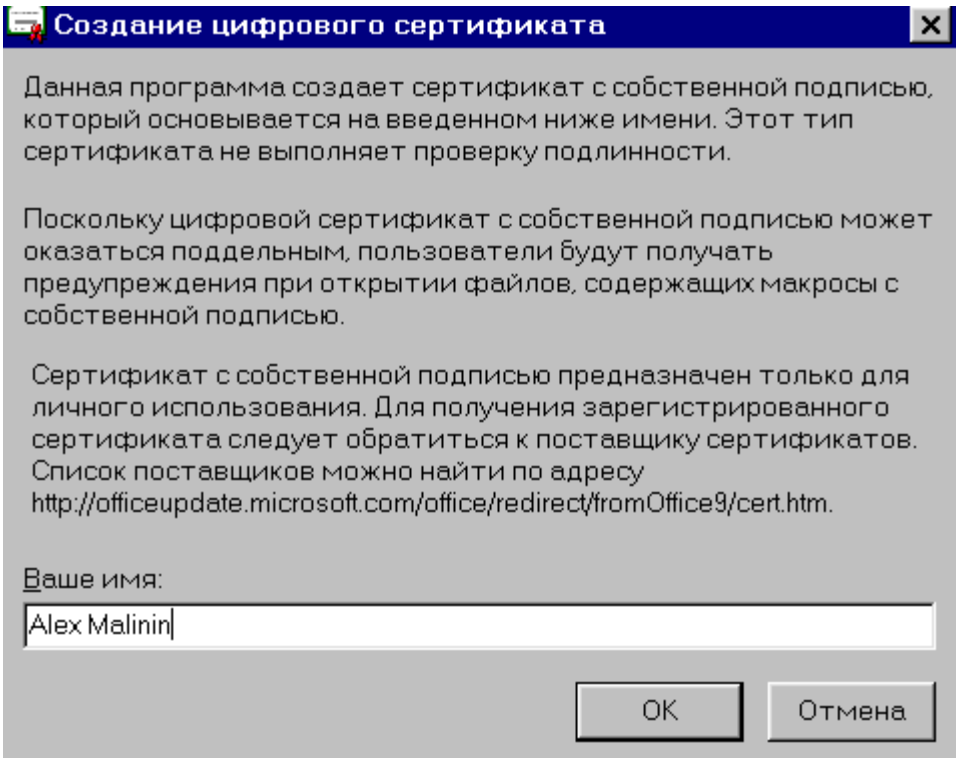


Рис. 424-1

Удаление же сертификата выполняется довольно хитрым способом, с помощью (кто бы мог ожидать?) Internet Explorer. Для этого в IE нужно выбрать команду "Сервис|Свойства обозревателя", затем выделить вкладку "Содержание", нажать кнопку "Сертификатов...", открыть вкладку "Личные" в окне "Диспетчер Сертификатов" и уже там проводить операции с сертификатами (рис. 424-2).

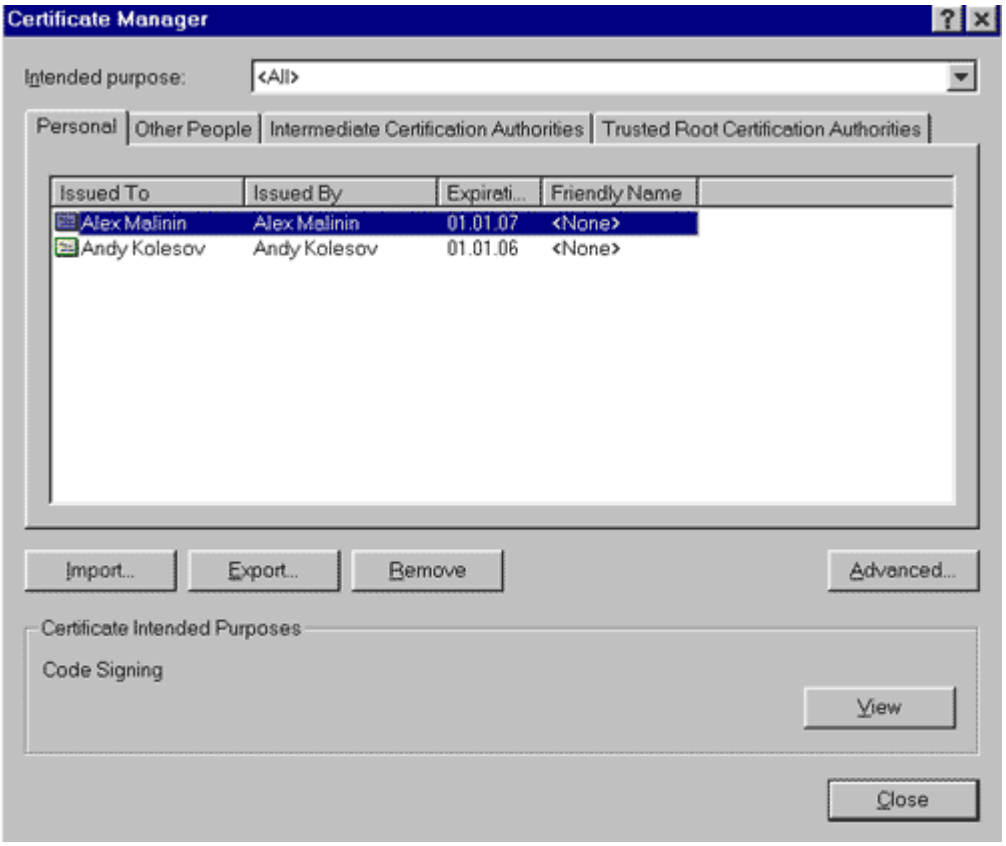


Рис. 424-2

2. **Подключение цифровой подписи.**

Электронная подпись подключается к VBA-проекту в среде VBA с помощью команды Tools|Digital Signature, которая выводит соответствующее окно "Цифровая подпись" (рис. 424-3).

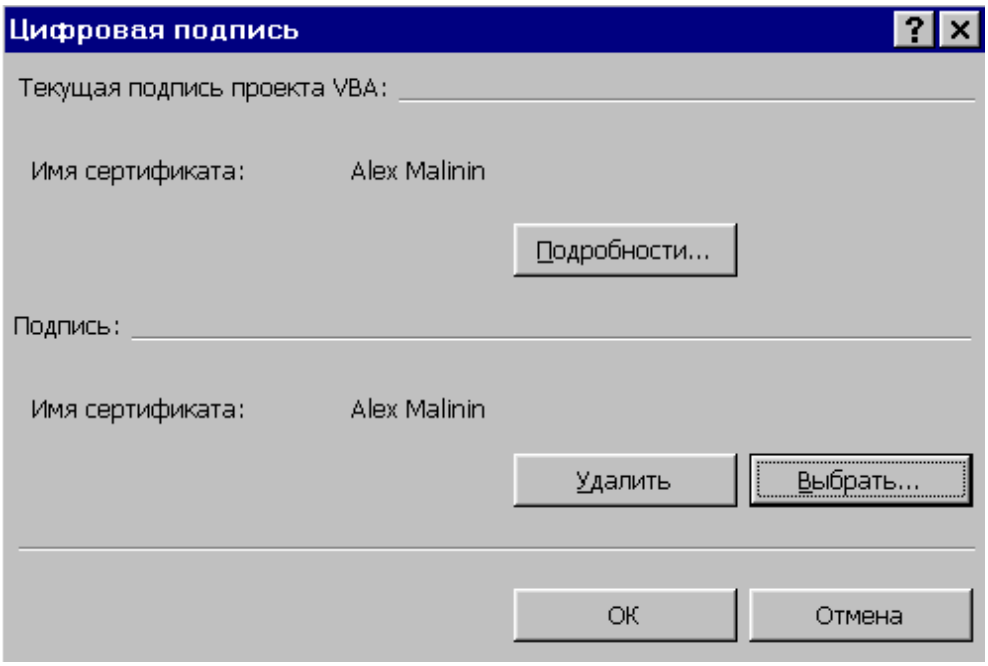


Рис. 424-3

Для выбора нужного сертификата нужно нажать кнопку "Выбрать", после чего выдается окно Select Certificate со списком имеющихся сертификатов (рис. 424-4).

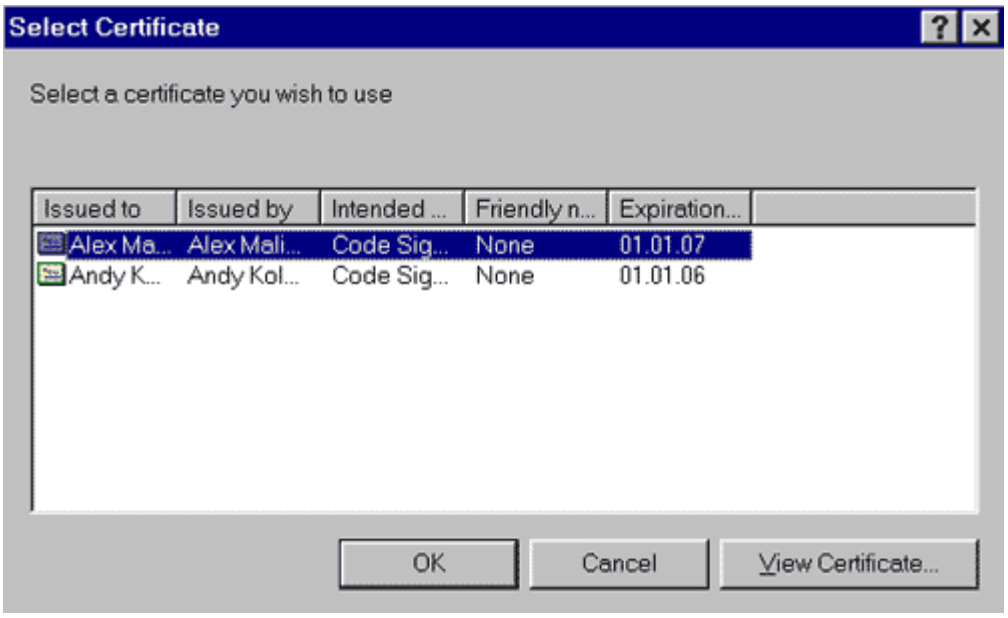


Рис. 424-4

Нажав кнопку, можно ознакомиться с более детальной информацией о сертификате в окне Certificate с тремя вкладками (рис. 424-5).

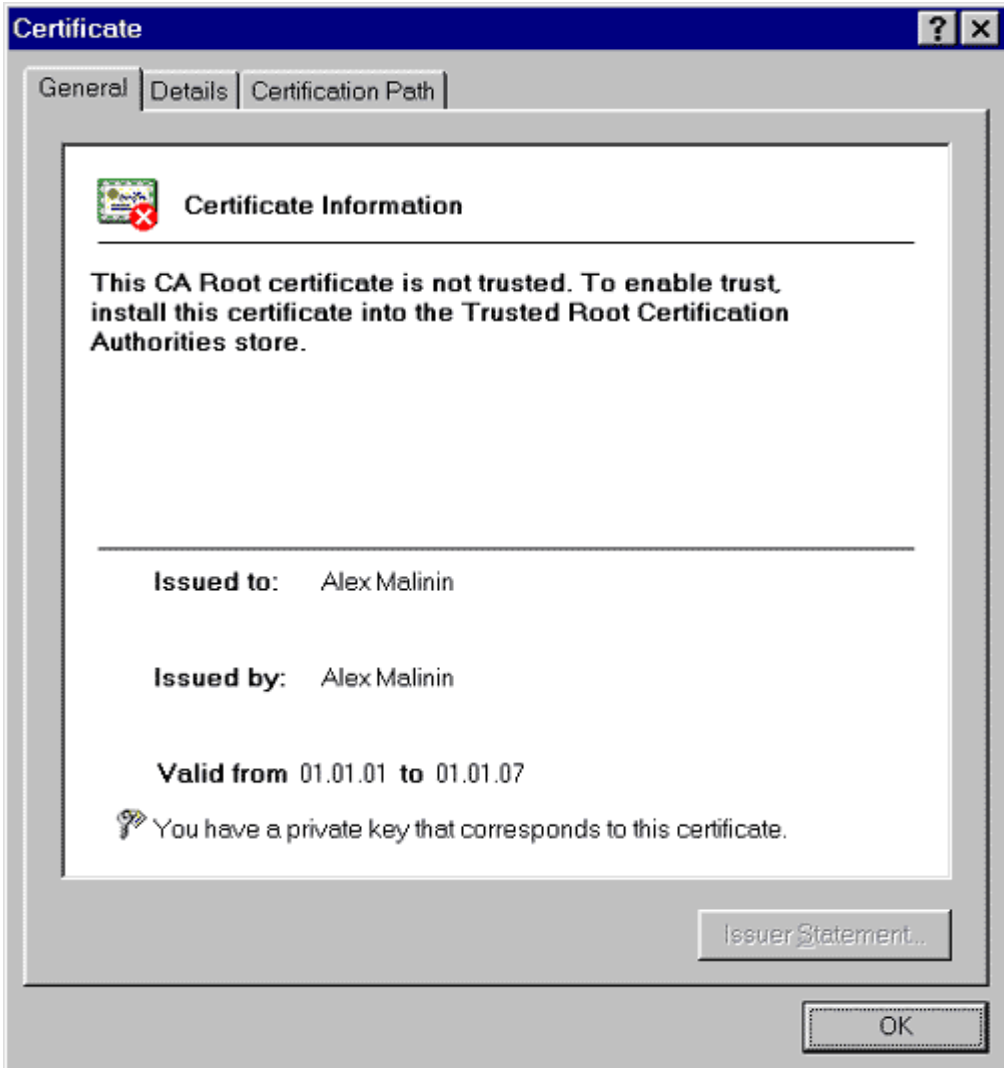


Рис. 424-5

К этой же информации можно получить доступ из других диалоговых окон при работе с сертификатами (обычно они называются "Подробности"). При желании можно в любой момент поменять подпись проекта или удалить ее совсем.

3. **Признание подписи в качестве надежной.**

   Эта операция выполняется только в момент загрузки проекта с наличием макросов, имеющих подписи, в режиме среднего и высокого уровней безопасности.

   В этом случае выдается окно предупреждения (рис. 424-6).

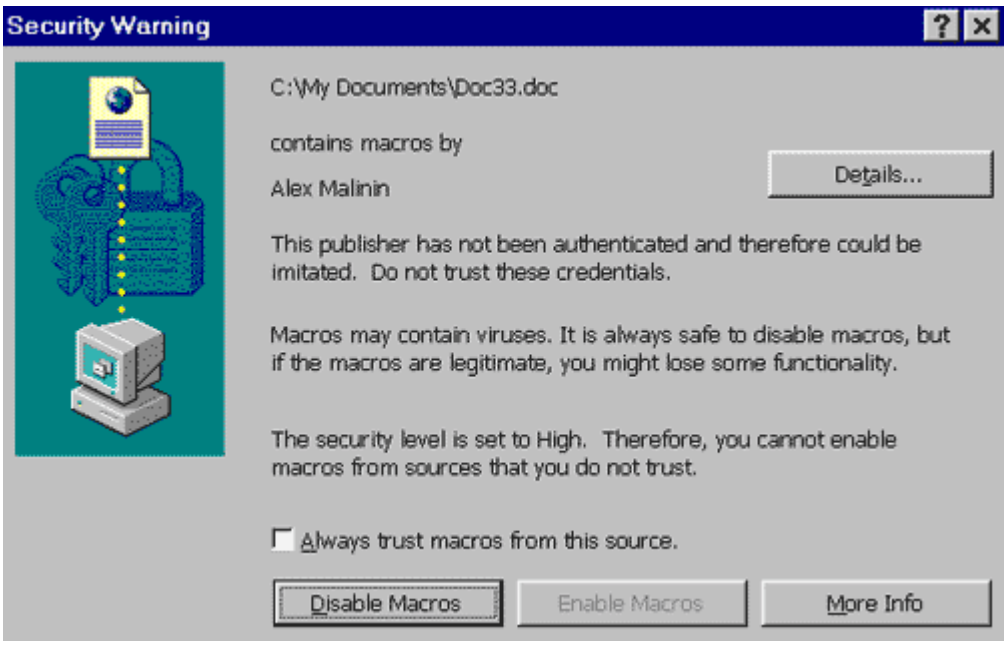


Рис. 424-6

В этот момент можно посмотреть более детальную информацию о данном сертификате (кнопка Details). Если вы считаете данный источник надежным, то нужно установить флажок Always trust macros from this source [Всегда доверять макросам этого источника] — и данный сертификат автоматически попадет в список надежных. Эти можно увидеть во вкладке Trusted Source окна Security (рис. 424-7). Здесь же можно удалить ненужные подписи.

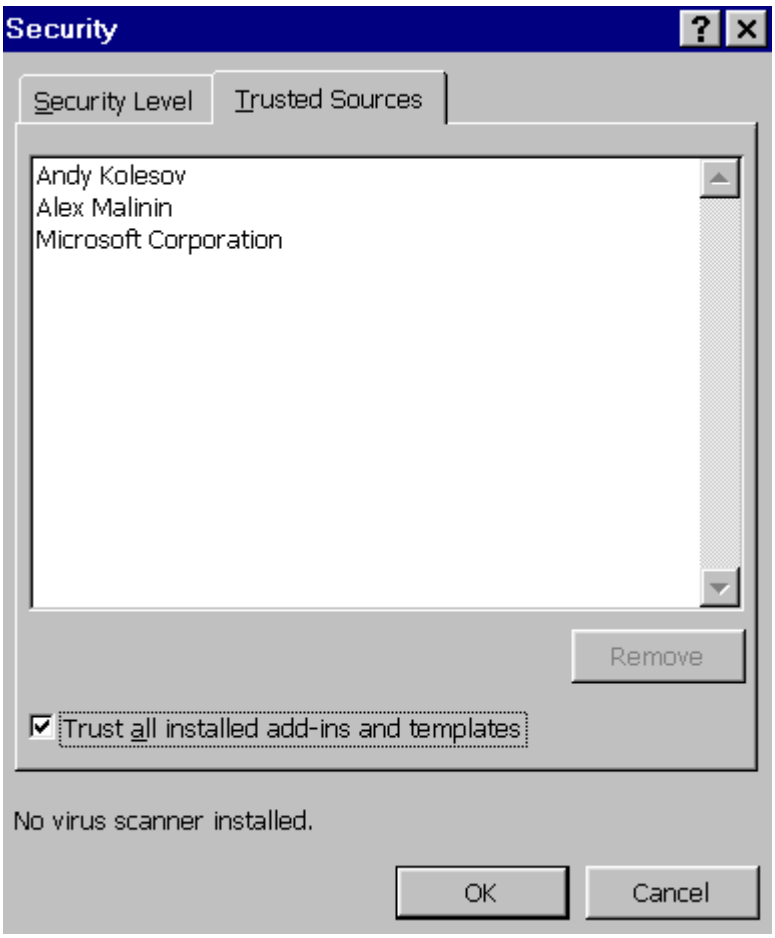


Рис. 424-7

Обратите внимание, что при использовании среднего уровня безопасности вы можете разрешить использование макросов проекта даже без занесения подписей в список доверенных (в том числе загружать проекты без подписей, например созданных в Office 97). При работе с высоким уровнем это можно сделать, только признав подпись (установив флажок "Всегда доверять"), то есть макросы проектов без подписей вообще нельзя использовать.

Замечание для Office 2000

Следует обратить внимание еще на один важный момент, который имел место в приложениях Office 2000. Порой здесь возникает ситуация, когда пользователь не может сохранить проект (документ с макрокодом) с электронной подписью — запись документа возможна только без подписи. При этом выдается неверная диагностика о нехватке места на диске.

Причина такого поведения — наличие внутренних синтаксических ошибок в коде проекта. Поэтому при обнаружении подобного сообщения о невозможности сохранения документа с электронной подписью, прежде чем отменять подпись, рекомендуем вам проверить работоспособность вашего кода. Довольно часто ошибка связана с отсутствием описания переменной или ссылки на внешний объект. Чтобы лучше понять суть ситуации, сделайте следующий простой пример в Word: создайте новый документ, перейдите в среду VBA и там создайте макрокоманду Test1:

```
Sub Test1 ()
```

```
    Avar = 1
End If
```

Разумеется, сначала должен быть задан режим Option Explicit (обязательное объявление переменных).

Теперь установите электронную подпись и попробуйте сохранить документ. Скорее всего, у вас появится сообщение о нехватке места на диске для записи файла. Запустите макрокоманду Test1 на выполнение — транслятор выдаст сообщение о синтаксической ошибке (не определена переменная Avar). Добавьте в процедуру описание:

```
Dim Avar As Integer
```

Теперь макрокоманда станет выполняться (правда, не делая ничего полезного), и документ тоже без проблем сохранится с электронной подписью.

Судя по всему, такое поведение Word 2000 является ошибкой (об этом говорит хотя бы выдача неверной диагностики). В Word 2002 эта ситуация исправлена — сохранение проекта с подписью выполняется независимо от ошибок кода.

Совет 425. Идентификация элементов управления

Порой бывает необходимо узнать внутри программного кода тип элемента управления, с которым выполняется в данный момент работа. Например, вы хотите изменить текст на всех командных кнопках формы или изменить какие-то свойства элементов управления определенного типа. Для подобной идентификации лучше всего подходит оператор TypeOf, который может использоваться в операторе If...Then:

```
If TypeOf ctl Is CommandButton Then
  ' Выполнить что-то
End If
```

Для идентификации типа произвольного элемента управления можно использовать, например, такую процедуру:

```
Private Sub Command1_Click()
  Dim ctl As Control
  Dim str As String
  For Each ctl In Me.Controls
    If TypeOf ctl Is CommandButton Then _
      str = "CommandButton"
    If TypeOf ctl Is TextBox Then str = "TextBox"
    If TypeOf ctl Is OptionButton Then str = _
      "OptionButton"
    If TypeOf ctl Is DriveListBox Then str = _
      "DriveListBox"
    If TypeOf ctl Is DataCombo Then str = _
      "DataCombo"
    MsgBox "Элемент управления типа " & str
  Next ctl
End Sub
```

Совет 426. Предотвращение многократного запуска VB-программы на одном компьютере

Возможно, вам необходимо предотвратить возможность повторного запуска вашего приложения (если оно уже запущено). Для этого можно воспользоваться свойством PrevInstance объекта App, например таким образом:

```
Private Sub Main()
  If App.PrevInstance Then
    ' приложение уже запущено
    Exit Sub ' завершить процедуру (а значит и приложение)
  End If
  ' продолжение работы программы
End Sub
```

Совет 427. Удостоверьтесь, что используете нужный провайдер при создании структурированных наборов данных ADO

Как вы, возможно, знаете, технология ADO позволяет включать иерархические структуры данных в один объект Recordset. Однако при создании такого объекта нужно правильно указать опции провайдера. Например, типичный вариант формирования стандартного набора данных может использовать такую строку соединения:

```
"Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\MyData\SomeDB.mdb"
```

Но для создания структурированного объекта Recordset вам нужно в качестве провайдера использовать SDataShape. К тому же вы должны указать, какой механизм OLEDB будет обеспечивать поддержку данных:

```
"Provider=MSDataShape;Data Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=C:\MyData\SomeDB.mdb"
```

или в случае SQL Server users:

```
"Provider=MSDataShape;Data Provider=SQLOLEDB;Data Source=MyServerName;Initial
Catalog=SomeSQLDB"
```

Кроме того, можно указать подобную строку в свойстве Provider объекта ADO Connection.

Совет 428. Программная реализация копирования экрана

Наш читатель Евгений спрашивает, как программным образом реализовать режим PrintScreen. А спустя пару дней сам же прислал свое решение этой задачи:

```
Private Declare Function BitBlt Lib "gdi32" _
   (ByVal hDestDC As Long, ByVal x As Long, _
    ByVal y As Long, ByVal nWidth As Long, _
    ByVal nHeight As Long, ByVal hSrcDC As Long, _
    ByVal xSrc As Long, ByVal ySrc As Long, _
    ByVal dwRop As Long) As Long
Private Declare Function GetDesktopWindow _
    Lib "user32" () As Long
Private Declare Function GetDC _
    Lib "user32" (ByVal hwnd As Long) As Long
Private Declare Function ReleaseDC Lib "user32" _
```

```
    (ByVal hwnd As Long, ByVal hdc As Long) As Long
Private Const SRCCOPY = &HCC0020
'_____________________________________________
Private Sub Command1_Click()
  Dim hwndScreen As Long
  Dim hScreenDC As Long
  Dim Res As Long
  hwndScreen = GetDesktopWindow()
  hScreenDC = GetDC(hwndScreen)
  Res = BitBlt(hdc, 0, 0, ScaleWidth, ScaleHeight, hScreenDC, 0, 0, SRCCOPY)
  Res = ReleaseDC(hwndScreen, hScreenDC)
  Form1.WindowState = vbMaximized 'распахиваем окно на полный экран
End Sub
```

Но только для формы нужно установить свойство AutoRedraw=True.

Совет 429. Как открыть ниспадающее меню программным образом

Как известно, VB 6 включает возможность задания Style=5 (tbrDropDown) для объекта Button элемента управления Toolbar. В этом случае можно добавить кнопке несколько объектов ButtonMenu. Но открыть меню можно только с помощью щелчка стрелки. То есть нельзя выполнить такую операцию из программного кода. Эта задача может быть решена с помощью следующего кода:

```
Private Type POINTAPI
  x As Long
  y As Long
End Type
Private Declare Function GetCursorPos Lib "user32" _
  (lpPoint as POINTAPI) As Long
Private Declare Function ClientToScreen Lib "user32" _
  (ByVal hWnd As Long, _
   lpPoint as POINTAPI) As Long
Private Declare Function SetCursorPos Lib "user32" _
  (ByVal x As Long, ByVal y As Long) As Long
Private Declare Function ShowCursor Lib "user32" _
  (ByVal bShow As Long) As Long
Private Declare Sub mouse_event Lib "user32" _
  (ByVal dwFlags As Long, ByVal dx As Long, _
   ByVal dy As Long, ByVal cButtons As Long, _
   ByVal dwExtraInfo As Long)
Private Sub ShowPopUpMenu _
  (TB As Toolbar, IndexOfButton%)
  ' раскрываем меню
  CONST MOUSEEVENTF_LEFTDOWN = &H2
  Const MOUSEEVENTF_LEFTDOWN = &H2
  Const MOUSEEVENTF_LEFTUP = &H4
  Dim Pt As POINTAPI, oldPt As POINTAPI
  With TB.Buttons(IndexOfButton)
    If Not (.Style = tbrDropdown Then Exit Sub
    ' режим разрешен
    Call GetCursorPos(oldPt) ' запоминаем позицию курсора
    Call ClientToScreen(TB.hWnd, Pt)
    Call ShowCursor(False)
    ' устанавливаем курсор на стрелку
    Call SetCursorPos(Pt.x + ((.Left + .Width) / _
       Screen.TwipsPerPixelX) - 1, _
       Pt.y + ((.Top + .Height)\ 2 /_
       Screen.TwipsPerPixelY))
  End With
  ' имитируем щелчок мышью
```

```
  Call mouse_event(MOUSEEVENTF_LEFTDOWN, _
     0, 0, 0, 0)
  Call mouse_event(MOUSEEVENTF_LEFTUP, _
     0, 0, 0, 0)
  ' восстанавливаем позицию курсора мыши
  Call SetCursor(oldPt.x, oldPt.y)
  Call ShowCursor (True)
End Sub
```

Используя эту подпрограмму, можно, например, открыть меню, когда пользователь щелкнет саму кнопку:

```
Sub ToolBar1_ButtonClick _
   (ByVal Button As MSComctllib.Button)
  Call ShowPopUpMenu (Toolbar1.Button.Index)
End Sub
```

Совет 430. Формирование кода цвета для HTML

Эта функция преобразует числовое значение цвета в строковую переменную для использования в HTML:

```
Public Function HtmlHexColor _
  (ByVal ColorValue As Long) As String
  Dim r As Byte
  Dim g As Byte
  Dim b As Byte
  ' преобразование цвета (если это нужно)
  Call OleTranslateColor _
    (ColorValue, 0&, ColorValue)
  r = ColorValue Mod &H10
  g = (ColorValue \ &h100) Mod &H100
  b = (ColorValue \ &h10000) Mod &H100
  HtmlHexColor = "#" & _
    Right$("0" & Hex$(r),2) & _
    Right$("0" & Hex$(g),2) & _
    Right$("0" & Hex$(b),2)
End Function
```

Совет 431. Модернизация объекта Err

Как это ни странно, но вы можете расширить функциональность встроенного объекта Err. Создайте класс с именем Cerror и включите в него такую процедуру свойства:

```
Public Property Get Number() As Long
  Number = VBA.Err.Number
End Property
Далее создайте BAS-модуль со следующим кодом:
Public Function Err() As Cerror
  Static oErr As Cerror

  If oErr Is Nothing Then
    Set oErr = New Cerror
  End If
  Set Err = oErr
End Function
```

Теперь вы можете обращаться к созданному объекту Cerror для которого можно создать любой набор свойств и методов.

Андрей Колесов, Ольга Павлова

© Андрей Колесов, Ольга Павлова, 2002
Авторский вариант. Статья была опубликована с незначительной литературной правкой в журнале "КомпьютерПресс" 04/2002, компакт-диск.

Совет 432. Как позволить пользователям держать командную кнопку в нажатом состоянии

Командные кнопки достаточно часто используются в VB-проектах, но, как известно, они реагируют только на однократный щелчок мыши. Порой возникают ситуации, когда полезно реализовать режим нажатия и удержания в этом состоянии кнопки в целях повторного выполнения в течение этого времени какого-то программного кода. Возможно, готовый вариант такого режима появится когда-нибудь в будущем, а пока мы покажем, как его можно реализовать уже сегодня с помощью элемента управления Timer.

В сущности, когда пользователь щелкает на кнопке, Timer может выполнять повторные обращения к какому-то программному коду. Если же пользователь отпустит кнопку, таймер отключится и прекратит выполнять обращения. Для осуществления нашей цели воспользуемся событиями MouseDown и MouseUp командной кнопки. Чтобы посмотреть, как это может быть сделано, откройте стандартный VB-проект, разместите на форме метку, командную кнопку и таймер и напишите для них следующий код:

```
Private Sub Command1_MouseDown(Button As Integer, _
  Shift As Integer, X As Single, Y As Single)
  ' при нажатии кнопки запускаем таймер
  Timer1.Enabled = True
End Sub

Private Sub Command1_MouseUp(Button As Integer, _
  Shift As Integer, X As Single, Y As Single)
  ' при отжатии кнопки отключаем таймер
  Timer1.Enabled = False
End Sub

Private Sub Form_Load()
  ' в исходном состоянии таймер отключен
  Timer1.Enabled = False
  ' интервал работы таймера 2 с
  Timer1.Interval = 2000
End Sub

Private Sub Timer1_Timer()
  ' периодически выполняемый код
  ' с заданным интервалом времени
  Label1.Caption = Now
End Sub
```

Теперь запустите проект, нажмите и удерживайте кнопку. Вы увидите, что в метке будут выдаваться значения текущего времени с обновлением каждые две секунды. Когда вы отпустите кнопку, отсчет времени прекратится.

Совет 433. Преобразование текстовых записей в массив с помощью Split

Довольно много VB-программ работает с данными, вводимыми непосредственно из обычных последовательных текстовых файлов. При этом исходная информация часто представлена в виде записей, поля которых было бы полезно преобразовать в массив. В этом случае можно воспользоваться функцией Split, реализованной в VB 6.0.

Как известно, Split позволяет преобразовать отдельные поля строки (разделенные каким-либо символом) в массив типа Variant array. По умолчанию Split использует пробел в качестве разделителя. Соответственно следующий код:

```
Dim vArray As Variant
vArray = Split("This is some function!")
```

создаст массив (с нижним индексом, который равен нулю!) из четырех элементов, и в каждом из них будет записано одно слово.

При желании можно указать любой другой разделитель, например Tab:

```
Split(someWildString, vbTab)
```

Довольно часто в качестве разделителя используется запятая. Предположим, ваш исходный файл содержит адреса людей, записанные примерно в таком формате:

```
Fred, 10001, 28 Flat St., Bedrock, UI
```

Программный код обработки таких исходных данных может выглядеть примерно следующим образом:

```
Private Sub Command1_Click()
  Dim Record As String
  Dim aryRecord As Variant

  Open "C:\bedrock.txt" For Input As #1
  Line Input #1, Record
  aryRecord = Split(Record, ",")
  ' распечатка отдельных полей
  For Each itm In aryRecord
    Debug.Print itm
   Next itm
  Close #1
End Sub
```

Совет 434. Как выгрузить VB-форму нажатием Esc

Эту задачу можно очень просто решить с помощью такого кода:

```
Private Sub Form_KeyPress(KeyAscii As Integer)
  If KeyAscii = 27 Then
   Unload Me
  End If
End Sub
```

Совет 435. Как определить относительный адрес файла

Иногда необходимо использовать адрес файла или каталога относительно какого-то другого адреса. Например, вы хотите узнать адрес файла с именем D:\TMP\MyFile.txt относительно текущего каталога D:\MYCAT\ (результат должен быть "..\TMP\MyFile.txt"). Это можно сделать с помощью API-функции из библиотеки SHLWAPI (Shell Light Weight API), которая поставляется в составе Internet Explorer 4.0 и старше:

```
Private Declare Function PathRelativePathToW _
    Lib "shlwapi.dll" (ByVal pszPath As Long, _
    ByVal pszFrom As Long, ByVal dwAttrFrom As Long, _
    ByVal pszTo As Long, ByVal dwAttrTo As Long) _
    As Boolean

Private Function GetRelativePath _
  (ByVal sPathFrom As String, _
   ByVal sPathTo As String) As String
   ' Определение относительного адреса
   ' каталога или файла
   Dim sRelativePath As String
   sRelativePath = Space(260) ' резервируем буфер
   '
   If PathRelativePathToW(StrPtr(sRelativePath), _
             StrPtr(sPathFrom), vbDirectory, _
             StrPtr(sPathTo), 0) Then  ' определили адрес
             MsgBox sRelativePath
     GetRelativePath = Left(sRelativePath, _
        InStr(sRelativePath, vbNullChar) - 1)
   Else  '
     GetRelativePath = "*"
   End If
End Function

Private Sub Command1_Click()
  ' txtFromPath должен содержать путь каталога ОТКУДА
  ' txtToPath должен содержать путь файла КУДА
  ' txtRelativePath будет содержать относительный путь КУДА
  '
  txtRelativePath.Text = GetRelativePath( _
    txtFromPath.Text, txtToPath.Text)
  If txtRelativePath.Text = "*" Then
    txtRelativePath.Text = "Ошибка"
  End If
End Sub
```

Совет 436. Использование системных значков в стиле MsgBox

Вы можете разместить на свою форму стандартные значки, используемые при выводе окна MsgBox, с помощью следующего кода:

```
Private Enum StandardIconEnum
  ' константы для определения вида окна
  IDI_ASTERISK = 32516&    ' как vbInfomation
  IDI_EXCLAMATION = 32515& ' как vbExclamation
  IDI_QUESTION = 32515& ' как vbQuestion
  IDI_HAND = 32513& ' как vbCritical
End Enum

Private Declare Function LoadStandardIcon _
  Lib "user32" Alias "LoadIconA" _
```

```
  (ByVal hInstance As Long, _
  ByVal lpIconName As StandardIconEnum) As Long
Private Declare Function DrawIcon Lib "user32" _
  (ByVal hdc As Long, ByVal x As Long, ByVal y As Long, _
   ByVal hIcon As Long) As Long

Private Sub Form_Paint()
   Dim hIcon As Long
   hIcon = LoadStandardIcon(0&, IDI_HAND)
   Call DrawIcon(Me.hdc, 10&, 10&, hIcon)
End Sub
```

В качестве функции LoadStandard мы использовали обычный прототип API-функции LoadIcon, для которой для последнего аргумента вместо типа Long задано перечисление StandardIconEnum.

Совет 437. Как получить описание файла

Нижеследующая процедура позволяет получить описание заданного файла, которое вы можете видеть в Windows Explorer в поле Type:

```
Declare Function SHGetFileInfo _
  Lib "shell32.dll" Alias "SHGetFileInfoA" _
  (ByVal pszPath As String, _
  ByVal dwFileAttributes As Long, _
  psfi As SHFILEINFO, _
  ByVal cbFileInfo As Long, _
  ByVal uFlags As Long) As Long

Const SHGFI_TYPENAME = &H400

Type SHFILEINFO
  hIcon As Long
  iIcon As Long
  dwAttributes As Long
  szDisplayName As String * 260
  szTypeName As String * 80
End Type

Function GetFileType(lpStrFile As String) As String
  Dim sfi As SHFILEINFO
  '
  ' Обращение к API-функции для заполнения структуры
  ' данными с описанием файла
  If SHGetFileInfo(lpStrFile, 0, sfi, _
     Len(sfi), SHGFI_TYPENAME) Then
     ' возвращен тип файла
     GetFileType = Left$(sfi.szTypeName, _
       InStr(sfi.szTypeName, vbNullChar) - 1)
  Else
    ' если ошибка, то пишем:
    GetFileType = "Неизвестный файл"
  End If
End Function
```

Совет 438. Создание MDI-формы без строки Caption

Возможно, вы захотите выдать MDI-форму без строки заголовка и соответственно безо всех размещенных на нем кнопок. Это можно легко сделать посредством обращения к

нескольким API-функциям. Обратите внимание: хотя системное меню не будет видно, к его командам можно обращаться с помощью горячих клавиш, например Alt-F4 или Alt-Space. Если вы хотите блокировать обращение к системному меню, то "раскомментарьте" строку Xor WS_SYSMENU в нижеследующем коде (но имейте в виду, что Alt-F4 все равно будет работать):

```
Declare Function GetWindowLong _
  Lib "user32" Alias "GetWindowLongA" _
  (ByVal hwnd As Long, ByVal nIndex As Long) As Long

Declare Function SetWindowLong _
  Lib "user32" Alias "SetWindowLongA" _
  ByVal hwnd As Long, ByVal nIndex As Long, _
  ByVal dwNewLong As Long) As Long

Const GWL_STYLE = (-16)
Const WS_CAPTION = &HC00000
Const WS_SYSMENU = &H80000

Private Sub MDIForm_Load ()
  Dim lStyle As Long
  lStyle = GetWindowLong(Me.hWnd, GWL_STYLE)
  ' блокировка вывода строки заголовка:
  lStyle = lStyle Xor WS_CAPTION
  ' уберите комментарий, чтобы блокировать системное меню:
  ' lStyle = lStyle Xor WS_SYSMENU
  Call SetWindowLong(Me.hWnd, GWL_STYLE, lStyle)
  '  Для VB4/32 нужно выполнить еще две строки кода:
  ' Form1.Show
  ' Unload Form1
End Sub
```

Если вы работаете с VB 4.0 (с 32-разрядной версией), то нужно обязательно загрузить дочернюю форму перед тем, как убрать строку заголовка.

Совет 439. Как преобразовать цветное изображение в черно-белое (Grayscale)

Если вы хотите представить цветное избражение в виде черно-белого (точнее, с серой цветовой шкалой), то можно использовать следующую функцию для преобразования кода цвета:

```
Public Function GrayScale (ByVal Colr As Long) As Long
  ' Преобразование кода цветного изображения (3 байта)
  ' в серый (байт) с кодом от 0 (черный) до 255 (белый)
  Dim R&, G&, B&
  ' разложение цветового кода на составляющие
  R = Colr Mod 256  ' красный
  Colr = Colr \ 256
  G = Colr Mod 256  ' зеленый
  Colr = Colr \ 256
  R = Colr Mod 256  ' синий
  ' Формирование серого эквивалента
  ' Это стандартный алгоритм, который используется
  ' для печати изображения цветного телевизора
  ' на черно-белый лазерный принтер
  GrayScale = (77 * R + 150 * G + 28 * B)/ 255
End Function
```

Совет 440. Как ограничить ввод для ComboBox

У стандартного текстового поля есть свойство MaxChars, позволяющее ограничить число вводимых символов. Раскрывающийся элемент управления ComboBox не имеет такого параметра, однако его можно легко реализовать с помощью простого вызова API-функции:

```
Private Declare Function SendMessage _
  Lib "user32" Alias "SendMessageA" _
  (ByVal hWnd As Long, _
   ByVal msg As Long, _
   ByVal wParam As Long, _
   ByVal lParam As Long) As Long
Private Const CB_LIMITTEXT = &H141
Private Form_Load ()
  Const Max_Char = 24 ' ограничить 24 символами
  Call SendMessage (Combo1.hWnd, _
       CB_LIMITTEXT, Max_Char, 0&)
End Sub
```

Совет 441. Копирование содержимого ListView в буфер обмена

Существует простая процедура, которая копирует содержимое элемента управления ListView, включая заголовки колонок, в буфер обмена для вставки, например в Excel или какое-то другое приложение:

```
Public Sub SendToClipboard (ByVal ListViewObj As MSComctlLib.ListView)

  Dim ListItemObj As MSComctlLib.ListItem
  Dim ListSubItemObj As MSComctlLib.ListSubItem
  Dim ColumnHeaderObj As MSComctlLib.ColumnHeader
  Dim ClipboardText As String
  Dim ClipboardLine As String

  ClipBoard.Clear
  '
  ' копирование заголовков:
  For Each ColumnHeaderObj In _
      ListViewObj.ColumnHeaders
    If ColumnHeaderObj.Index = 1 Then
      ClipboardText = ColumnHeaderObj.Text
    Else
      ClipboardText = ClipboardText & _
         vbTab & ColumnHeaderObj.Text
    End If
  Next
  ' содержимое колонок:
  For Each ListItemObj In _
      ListViewObj.ListItems
    ClipboardLine = ListItemObj.Text
    ' содержимое подчиненных элементов
    For Each ListSubItemObj In _
        ListItemObj.ListSubItems
      ClipboardLine = ClipboardLine & _
         vbTab & ListSubItemObj.Text
    Next
    ClipboardText = ClipBoardText & vbCrLf & _
      ClipboardLine
  Next
  Clipboard.SetText ClipboardText
```

```
End Sub
```

Совет 442. Центровка формы с учетом системных панелей

Довольно часто программисты стараются выводить форму по центру экрана. Однако обычно они не учитывают того, что на экране уже находятся разные визуальные элементы, такие, например, как панель задач или панель MS Office. Приведенная ниже функция позволяет определить местоположение формы по центру только того места экрана, которое действительно свободно:

```
Private Declare Function GetSystemMetrics _
   Lib "user32" (ByVal nIndex As Long) As Long

Private Declare Function GetWindowLong _
  Lib "user32" Alias "GetWindowLongA" _
  (ByVal hwnd As Long, ByVal nIndex As Long) As Long

Private Const SM_CXFULLSCREEN = 16
Private Const SM_CYFULLSCREEN = 17
Public Sub CenterForm(myForm As Form)
  ' Центровка формы на экране
  ' с учетом системных панелей
  Dim Left As Long, Top As Long
  Left = Screen.TwipsPerPixelX * _
         GetSystemMetrics(SM_CXFULLSCREEN) / 2 - _
         myForm.Width / 2
  Top = Screen.TwipsPerPixelY * _
         GetSystemMetrics(SM_CYFULLSCREEN) / 2 - _
         myForm.Height / 2
  myForm.Move Left, Top
End Sub

Private Sub Form_Load()
 Call CenterForm(Me)
End Sub
```

Совет 443. Программная имитация щелчка мышью

Если вы хотите программным образом имитировать щелчок мышью на каком-то элементе управления, то для этого следует воспользоваться API-функцией SendMessage, для которой второй параметр должен быть равен BM_CLICK = &HF5, а два следующие — нулю. Если элемент управления, который вы хотите щелкнуть, будет некоторое время отрабатывать этот щелчок, то следует выполнить асинхронный щелчок с помощью функции PostMessage. Для проверки этого алгоритма выполните такой тестовый пример:

```
Private Declare Function SendMessage _
  Lib "user32" Alias "SendMessageA" _
  (ByVal hwnd As Long, ByVal wMsg As Long, _
  ByVal wParam As Long, lParam As Any) As Long
Private Declare Function PostMessage _
  Lib "user32" Alias "PostMessageA" _
  (ByVal hwnd As Long, ByVal wMsg As Long, _
  ByVal wParam As Long, lParam As Any) As Long
Private Const BM_CLICK = &HF5

Private Sub Command2_Click()
    Debug.Print "Начало Command1_Click"
    Call SendMessage(Check1.hwnd, BM_CLICK, 0, ByVal 0&)
    Call SendMessage(Option1.hwnd, BM_CLICK, 0, ByVal 0&)
```

```
    Call SendMessage(Command1.hwnd, BM_CLICK, 0, ByVal 0&)
    Debug.Print "Конец Command1_Click"
End Sub

Private Sub Option1_Click()
    Debug.Print "Option1_Click"
End Sub

Private Sub Check1_Click()
    Debug.Print "Check1_Click"
End Sub

Private Sub Command1_Click()
    Debug.Print "Command1_Click"
End Sub
```

Совет 444. Выполнение операций с помощью механизма Scripting

Мы уже писали ранее (см. советы 269[90], 384[91]) о возможности использования дополнительно элемента управления для выполнения скриптов в своем приложении (его можно загрузить по адресу http://msdn.microsoft.com/scripting/scriptcontrol). Однако его можно применять не только для обычных операций типа:

```
ScriptControl1.Eval("(2 * 3) + 5)")
```

На самом деле Microsoft Script Control может выполнять целые программы. Чтобы убедиться в этом, создайте текстовый файл, например с именем C:\Temp.txt, с помощью обычного Notepad, в котором запишите следующий код:

```
Sub Main ()
  MsgBox "Привет!"
End Sub
```

А затем выполните такой код:

```
Private Sub Command1_Click()
  Dim iFileNum As Long
  Dim sFileBuffer As String
  Dim sTemp As String
  iFileNum = FreeFile()  ' получение свободного номера
   ' ввод исходного кода и формирование строки
   Open "C:\Temp.txt" For Input As #iFileNum
   Do While Not EOF(iFileNum)
     Line Input #iFileNum, sTemp
     sFileBuffer = sFileBuffer & sTemp & vbCrLf
   Loop
   Close #iFileNum
   ' выполение операций:
   ScriptControl1.Reset
   ScriptControl1.AddCode sFileBuffer
   ScriptControl1.Run "Main"
End Sub
```

[90] http://www.visual.2000.ru/develop/ms-vb/tips/0004-2.htm
[91] http://www.visual.2000.ru/develop/ms-vb/tips/0106.htm

Заключительные советы тем, кто программирует на VB & VBA

Андрей Колесов, Ольга Павлова

© Андрей Колесов, Ольга Павлова, 2002
Авторский вариант. Статья была опубликована с незначительной литературной правкой в журнале "КомпьютерПресс" 06/2002, компакт-диск.

Кто не может сделать что-то сам, советует другим, как это лучше сделать. ***Из жизненного опыта авторов***

"Сериал" длиною в шесть лет

В начале 1996 года журнале"КомпьютерПресс" N 3/96 впервые была опубликована наша статья под названием "Советы для тех, кто программирует на Visual Basic"[92]. В ней мы давали всего три совета и, честно говоря, даже не предполагали, что потребуется продолжение. И уж совсем не думали, что эта любительская статья откроет "сериал", который растянется на шесть с гаком лет, а число советов приблизится к 450, и что с середины 1997 года параллельно с "Советами..." начнут публиковать наши тематические статьи о VB (это было связано с изучением возможностей новых версий VB), а еще спустя полгода — новый цикл публикаций под рубрикой "Разработка приложений в Office/VBA"[93].

Кстати, появление этого цикла статей именно в "КомпьютерПресс" было неслучайным: еще за четыре года до этого, в 1991 году, также в мартовском номере журнала появилась статья "QuickBASIC — это то, что вам нужно"[94], которая стала нашей первой публикацией в только зарождавшейся тогда отечественной компьютерной прессе.

Нужно открыть читателям наш маленький секрет. Дело в том, что до 1995 года мы были профессиональными программистами (профессиональными в том смысле, что именно этим ремеслом мы зарабатывали себе на жизнь) и работали с 1988 года в основном на Microsoft QuickBasic 4.5 for DOS (см. наш Web-ресурс "MS Basic/DOS"[95]). Хотя написание статей о разработке ПО в среде QB и не приносило особых доходов, но способствовало продвижению нашей "торговой марки", к тому же нам было просто интересно этим заниматься. Логичным продолжением нашей "писательской деятельности" стало то, что в 1995 году мы "завязали" с программированием и стали профессиональными (в том же смысле) журналистами, пишущими на "околокомпьютерные" темы, не связанные с техническими вопросами программирования.

Эту историю мы изложили, чтобы пояснить: мы никогда не занимались практическим программированием на VB. В конце 1994 года мы установили на свои компьютеры VB 3.0 только из любопытства, а первые Советы..." появились совершенно случайно. Таким образом, изучение VB/VBA в течение этих лет было связано исключительно с необходимостью написания все новых и новых советов и статей, так что его лучше

[92] http://www.visual.2000.ru/develop/ms-vb/tips/9603.htm
[93] http://www.visual.2000.ru/develop/ms-vb/tips/www.visual.2000.ru/develop/vb/compress/index.htm
[94] http://www.visual.2000.ru/kolesov/compress/cp9103.htm
[95] http://www.visual.2000.ru/develop/basicdos/index.htm

отнести к хобби, а не к профессиональной деятельности. Вступив "в бой", мы "сражались" более шести лет, пройдя путь от VB 3.0 до VB .NET (7.0).

Но всему на свете приходит конец. Судя по всему, эта публикация станет завершающей в затянувшемся "сериале". Основную причину этого можно сформулировать так: на хобби остается все меньше времени, а переход к описанию версии VB .NET потребует довольно значительных усилий.

Учитывая все вышесказанное, мы решили дать в этой завершающей статье **несколько советов для тех, кто собирается и в будущем программировать на Visual Basic**.

Совет 445. Изучайте VB в контексте его исторического развития

В современном VB, конечно же, очень сложно узнать его прародителя, созданного преподавателями Дартмутского колледжа (США) почти 40 лет назад, тем не менее связь времен легко прослеживается.

Долгие годы Бейсик был довольно примитивной системой, и о его использовании в качестве средства разработки серьезных программ не могло быть и речи. Однако нужно подчеркнуть, что Бейсик фактически никогда и не был языком программирования типа Фортрана или Алгола. Ведь он изначально представлял собой качественно новую технологию создания программ (с использованием режима "позднего связывания"!) в режиме интерактивного диалога между разработчиком и компьютером, то есть был прообразом современных систем быстрой разработки программ. Другое дело, что решение подобной задачи с помощью техники тех лет было возможно только за счет максимального упрощения языка программирования и использования транслятора типа "интерпретатор".

Сейчас вряд ли кто-то рискнет заявить, что сегодня Бейсик практически стал аналогом системы Microsoft Visual Basic, которая, в свою очередь, является самым популярным в мире инструментом разработки приложений. Конечно, у нее есть свои недостатки, многие относятся к ней с иронией, но факт остается фактом: не менее 80-90% программистов если не используют эту систему в работе, то, по крайней мере, знакомы с ней и при желании могут легко сотворить в ней что-то полезное. Существенный рост числа VB-программистов начался несколько лет назад, после появления MS Office 97/VBA — осваивать программирование в среде этого пакета стали многие его продвинутые пользователи.

Некоторые основные этапы развития Бейсика и Visual Basic приведены в таблице 1[96] и таблице 2[97]. Хотим похвастаться: на нашем "боевом" компьютере установлены почти все версии MS Basic для DOS и Windows, упомянутые в этих таблицах (кроме VB 1.0 и 2.0, VBA 1.0 и 4.0). Все они — лицензионные, причем версии до 1995 года выпуска (включая VB 3.0) мы покупали на свои деньги, позднее — получали "для изучения" от московского представительства Microsoft, за что ему очень благодарны.

К приведенной хронологии хотелось бы сделать еще несколько дополнительных комментариев:

[96] http://www.visual.2000.ru/develop/ms-vb/tips/table1.htm
[97] http://www.visual.2000.ru/develop/ms-vb/tips/table2.htm

1. Мы выделили в истории Бейсика три основных этапа, обозначенные в таблице 1[98]. К этому можно только добавить, что в силу исторических причин большинство российских программистов второй половины 80-х годов вообще не знало о возможностях современных Basic-систем той поры. Их представление о Бейсике в лучшем случае базировалось на знакомстве с весьма примитивным интерпретатором GW-Basic из состава MS DOS 4.0. Во многом именно такими устаревшими представлениями о Basic-системах объясняется очень настороженное отношение к VB российских программистов примерно до 1995 года.

   Успех же Visual Basic в США в значительной степени базировался на огромной популярности семейства Microsoft QuickBasic.

2. В конце 80-х годов насчитывалось около десятка систем Basic различных фирм-разработчиков. Однако главная борьба шла между QuickBasic (компания Microsoft) и TurboBasic (Borland). Вообще-то, конкуренция между этими двумя разработчиками средств программирования шла по целому спектру языков — Basic, Pascal и C. И результатом ее в 1989 году стало неявное мировое соглашение, когда Microsoft отказалась от дальнейшей поддержки Pascal, а Borland — Basic.

   В те времена многие эксперты, комментируя такой поворот событий, часто упоминали о личной заинтересованности руководителей Microsoft в Basic. Однако, как нам представляется, победа QuickBasic определялась чисто технологическими причинами — в этой системе была удачно реализована схема смешанного использования традиционных Basic-технологий и классических методов создания сложных программных систем.

   Turbo Basic имел много достоинств, главным же его недостатком был полный переход к компилятору, в результате чего он потерял свою изюминку — эффективную отладку программ.

3. В американском журнале PC Magazine от 28 сентября 1993 года был опубликован большой обзор рынка Basic-средств, где было проанализировано около десяти ведущих систем того времени, в том числе CA Realizer (Computer Associates), GFA-BASIC, TrueBASIC, PowerBASIC. От Microsoft были представлены три системы — PDS 7.1, VB/Win 3.0 и VB/DOS 1.0.

   Обзор этот весьма примечателен по двум причинам. Во-первых, системы для DOS и Windows рассматриваются в нем как равные конкуренты. Во-вторых, это был, кажется, последний случай, когда у Microsoft еще были реальные конкуренты.

   Любопытно, что выбор редакции в разделе Basic/DOS был сделан в пользу PowerBASIC 3.0, а не в пользу более сильного PDS 7.1 из-за "неясной позиции Microsoft в плане развития систем для DOS". Действительно, Microsoft темнила почти два года и даже выпустила совершенно тупиковый вариант VB/DOS, чтобы дождаться естественной кончины DOS'овского направления.

4. В конце 1993 года, одновременно с официальным объявлением о прекращении развития Basic/DOS, корпорация Microsoft объявила о намерении создать на основе VB новую универсальную систему программирования для прикладных программ, которая получила название Visual Basic for Applications (VB для приложений). Но нужно отметить, что, казалось бы, довольно ясная задача была реализована далеко не сразу.

[98] http://www.visual.2000.ru/develop/ms-vb/tips/table1.htm

Первый вариант VBA 1.0 появился в составе MS Office 4.0, но лишь в программах Excel 4.0 и Project 6.0. В других же приложениях — Word 6.0 и Access 2.0 — были собственные варианты Basic. К тому же VBA 1.0 довольно сильно отличался (причем он имел ряд существенных преимуществ) от используемой тогда универсальной системы VB 3.0.

Качественный перелом наступил в конце 1996 года — после выпуска MS Office 97, в котором была реализована единая среда программирования VBA 5.0, включенная в программы Word, Excel и PowerPoint. Более того, VBA 5.0 использовала тот же самый языковый механизм и среду разработки, что и универсальная система VB 5.0. В состав выпущенного два год назад пакета MS Office 2000 вошла соответственно версия VBA 6.0, используемая в шести программах — Word, Excel, PowerPoint, Access, Outlook, FrontPage.

Одновременно VBA активно продвигают в качестве отраслевого стандарта для управления программируемыми приложениями, объявив о возможности его лицензирования и выпустив соответствующий SDK.

5. Появление Visual Basic .NET открыло новый, четвертый, этап в истории Бейсика. В этом нет никаких сомнений.

Совет 446. Не спешите расставаться с VB 6.0

Выпуск VB .NET открывает качественно новый этап в развитии семейства VB. Хотя VB .NET имеет рабочий номер версии 7.0, но было бы правильнее назвать его VB .NET 1.0. Для VB-программистов миграция в .NET будет более сложной, чем переход от 16-разрядных версий Windows к 32-разрядным, не говоря уже о смене VB 5.0 на VB 6.0. Масштаб изменений скорее сравним с переходом с DOS Basic на Visual Basic.

Вывод из этого можно сделать следующий: две системы наверняка будут сосуществовать (как параллельно использовались DOS и Windows почти до середины 90-х годов), хотя сейчас время переходного периода, скорее всего, сократится до 2-3 лет.

Тут полезно вспомнить, что любой продукт практически всегда достигает "зрелости" (то есть его можно начинать использовать для решения критически важных бизнес-задач) лишь с выходом 3-й версии. Мы же сейчас имеем дело с версией .NET 1.0 и, несмотря на обилие рекламной шумихи, понимаем, что пока идет лишь этап практического изучения этой технологии.

Учитывая все вышеизложенное, было бы очень разумно, с точки зрения программистов, если бы Microsoft в течение некоторого времени продолжала продавать и поддерживать VB 6.0. Официальных заявлений на эту тему со стороны Microsoft пока нет, но, учитывая агрессивную стратегию бизнеса корпорации, можно не сомневаться, что Microsoft будет всячески форсировать переход разработчиков на VB .NET, точнее на Visual Studio .NET.

Совет 447. Помните о наших советах

Хотя наш "сериал" назывался "Советы тем, кто программирует на VB", на самом деле он был посвящен скорее программированию вообще, но на примере VB. А это означает, что большинство наших советов остаются актуальными — независимо от используемых версий языков, а в значительной степени и от самих средств программирования.

Попробуем кратко сформулировать основные наши рекомендации:

1. Всегда устанавливайте режим обязательного описания переменных (Option Explicit). Это делается автоматически, если вы установите флажок Require Variable Declaration в диалоговом окне Tools | Options | Editor.
2. Следите за правильным использованием типов данных. Избегайте неявного преобразование типов. Не используйте тип Variant, кроме случаев, когда это действительно необходимо. В VB .NET появилась дополнительная возможность жесткой проверки типов (запрет неявного преобразования типов данных, на необходимость которого мы неоднократно обращали внимание) — режим Option Strict On. Мы настоятельно рекомендуем включать его (вкладка Build диалогового окна Project Properties).
3. Используйте функции Windows API. Не менее 40% наших советов были связаны с применением Windows API, и одно это уже говорит о многом. На самом деле множество функций VB представляют собой выполненные по правилам синтаксиса VB обращения к Windows API, но довольно часто они реализуют лишь некоторые частные случаи использования этих функций, не говоря уже о том, что многие принципиальные возможности реализуются только через Windows API.

   Кроме многочисленных советов, теме Windows API посвящена наша статья "Работа с функциями Windows API и DLL"[99]. В своих публикациях мы часто упоминали (и еще чаще использовали в работе) серию книг американского автора и разработчика ПО Дана Эпплмана, посвященных Windows API для VB. Уже более восьми лет они неизменно входят в состав бестселлеров для VB-программистов. Год назад издательство "Питер" выпустило одну из последних книг этой серии: Эпплман Д. Win32 API и Visual Basic (+CD)/Пер. с англ. — СПб.: Питер, 2001. — 1120 с.: ил.[100]

   Настоятельно рекомендуем приобрести ее, поскольку это действительно книга из разряда "вечных" (хотя мы-то с вами понимаем, что "вечность" в ИТ — это срок более 4-5 лет).

4. Будьте очень аккуратны при работе со строковыми переменными. Помните, что это — динамические переменные и что операции с ними являются самыми ресурсоемкими. Советуем прочитать нашу статью "Особенности работы со строковыми переменными в VB"[101].
5. Учитывайте особенности работы VB-приложений в Windows с различными национальными установками. На эту тему можно посмотреть упомянутую статью "Особенности работы со строковыми переменными в VB"[102], а также "Особенности обработки дат в VB"[103].
6. Помните о различиях режимов раннего и позднего связывания. При прочих равных условиях рекомендуем применять раннее связывание. (Подробнее об этом см. в статье "Особенности технологий раннего и позднего связывания в VB"[104].)
7. VBA-программисты! Помните, что для освоения VBA вам крайне необходимо изучать обычный VB (в том числе наши советы и статьи о VB). Дело в том, что в многочисленной литературе по Office/VBA акцент обычно делается на особенности работы с объектной моделью того или иного офисного приложения. При этом аспекты собственно

[99] http://www.visual.2000.ru/develop/ms-vb/cp0009-2/vb-api.htm
[100] http://www.visual.2000.ru/kolesov/pcweek/2002/20517api.htm
[101] http://www.visual.2000.ru/develop/ms-vb/cp9910/strvb-11.htm
[102] http://www.visual.2000.ru/develop/ms-vb/cp9910/strvb-11.htm
[103] http://www.visual.2000.ru/develop/ms-vb/cp9907/90404-vb.htm
[104] http://www.visual.2000.ru/develop/ms-vb/cp0009-1/binding.htm

программирования детально не рассматриваются, так как авторы совершенно справедливо полагают, что они уже обстоятельно изложены в публикациях о VB. Таким образом, не менее 70% информации в статьях и книгах о VB являются актуальными для VBA (в отношении Windows API — 100%), но при этом в материалах о VBA эти сведения обычно приводятся в минимальном объеме.

Советуем также посмотреть нашу статью "Как осваивать программирование в среде MS Office/VBA 2000"[105].

Наверное, это далеко не все так называемые "общие" советы. Кстати, архив всех наших публикаций по VB/VBA можно найти на нашем Web-сайте[106].

Совет 448. Готовьтесь к VB .NET

Мы абсолютно уверены, что для тех, кто собирается продолжать заниматься программированием (а не как мы во времена перехода от DOS к Windows — сменить профессию), вообще не стоит вопрос о том, переходить или не переходить на VB .NET. Вопрос в другом — когда это случится?

Впрочем, еще одним реальным вариантом является переход с VB 6.0 на C#, обладающий более гибкими возможностями по сравнению с VB .NET. (Мы считаем, что Microsoft сохранила определенные функциональные различия между VB .NET и C# исключительно из маркетинговых соображений "разделяй и властвуй". Но при этом нужно подчеркнуть, что C# создавался с чистого листа, специально под .NET, а при модернизации VB .NET корпорация все же была вынуждена учитывать проблему унаследованного кода.) Но тут следует подумать: расширенные функции C# могут быть просто неактуальными для вас, а синтаксис VB выглядит, на наш взгляд, более четким и наглядным, по крайней мере привычным. Мы уверены, что для 99% VB-программистов функциональность VB .NET будет вполне достаточной. К тому же вы можете легко реализовать технологию смешанного программирования, работая на VB .NET и в случае необходимости создавая какие-то функции на C#.

Начать изучение VB .NET нужно было еще полтора года назад. Но и сейчас еще не поздно.

Итак, переход на VB .NET неизбежен. Когда же и как он произойдет?

В плане разработки новых приложений в среде .NET ситуация выглядит довольно прозрачно. Лучше начать с создания прототипов небольших некритичных приложений, но при этом желательно вести такие разработки для широкого круга задач, чтобы изучить технологию с разных сторон.

Не забывайте и о печальном опыте времен перехода от DOS к Windows: Многие программисты долго не хотели заниматься "тупым Windows-программированием" и в результате им пришлось менять профессию.

Но что делать с "критичными приложениями"? Основная проблема заключается в том, что при переходе с VB 6.0 на VB .NET нарушена совместимость кода, то есть будут возникать трудности при миграции приложений. В этом случае можно дать следующие советы:

[105] http://www.microsoft.ru/offext/officedev/articles/kolesov/
[106] http://www.visual.2000.ru/develop/vb/index.htm

- Разрабатывая сегодня приложения на VB 6.0, нужно знать о будущих проблемах миграции, чтобы минимизировать затраты по преобразованию кода. По этому поводу рекомендуем заглянуть на Web-адрес, где лежат наши публикации на тему "А ты готов к VB .NET"[107].
- В состав VB .NET входит Migration Wizard, который автоматически преобразует старый VB6-код. Но делает он это только на 95%, то есть в любом случае достаточно серьезную программу нужно будет просматривать, анализировать и корректировать вручную. А это уже очень серьезная работа, поэтому стоит подумать: стоит ли вообще выполнять миграцию? По оценкам компании Gartner, лишь 30-40% существующего сегодня VB-кода можно будет перевести на платформу .NET, остальное придется переписывать и перепроектировать. Не следует ожидать, что Microsoft будет серьезно озадачена созданием утилит для автоматического преобразования кода.

Таким образом, лучше всего уже сейчас определиться со своими текущими разработками, которые можно разделить на три категории:

1. жизненный цикл этих разработок будет определяться исключительно VB 6.0 (см. совет 446[108]);
2. разработки сегодня ведутся на VB 6.0 с перспективой перехода на VB .NET;
3. для новых приложений изначально выбран VB .NET.

Для последнего случая лучше всего подходят Web-приложения (VB .NET в этом плане имеет явные преимущества перед VB 6.0) и новые серверные приложения. Что касается клиентских приложений, то тут ключевым моментом является такой вопрос: установили ли уже ваши клиенты исполнительную среду .NET?

Мы не хотим в данной статье затрагивать тему новшеств VB .NET — этот вопрос требует отдельного рассмотрения. Ограничимся лишь одним простым примером:

> *Создайте в VB 6.0 новое приложение типа Standard. На форме разместите командную кнопку и напишите для нее такой код:*
>
> ```
> Private Sub Command1_Click()
>     Dim NowTime As Long
>     NowTime = Timer
>     MsgBox NowTime
> End Sub
> ```
>
> *Обратите внимание —это код всего модуля формы! А теперь загрузите этот проект в VB .NET и посмотрите, что у вас получилось в результате работы Migration Wizard (см. листинг[109]). Правда, блоки Windows Form Designer generated code и Upgrade Support являются чисто служебными (непонятно, зачем они вообще появляются в окне кода), но даже если их закрыть, то отличия в новой программе довольно заметны:*
>
> ```
> Option Strict Off
> Option Explicit On
> Imports VB = Microsoft.VisualBasic
> Friend Class Form1
>      Inherits System.Windows.Forms.Form
>
>     Private Sub Command1_Click _
> ```

[107] http://www.visual.2000.ru/develop/ms_net/index.htm
[108] http://www.visual.2000.ru/develop/ms-vb/tips/0206.htm
[109] http://www.visual.2000.ru/develop/ms-vb/tips/listing.htm

```
      (ByVal eventSender As System.Object, _
       ByVal eventArgs As System.EventArgs) _
       Handles Command1.Click
         Dim NowTime As Integer
         NowTime = VB.Timer()
         MsgBox(NowTime)
    End Sub
End Class
```

Обратите внимание, что по умолчанию VB .NET устанавливает режим Option Strict Off, а мы советуем вам обязательно выбирать On.

Вот еще одна рекомендация: имейте в виду, что при переходе на VB .NET вам придется изучать не столько сам язык, сколько архитектуру .NET. В связи с этим нужно понимать, что большинство публикаций, посвященных VB .NET или C#, на самом деле рассматривают именно архитектурные вопросы .NET, но на примере того или иного языка (разница между которыми не очень значительна). Незадолго до написания этой статьи мы познакомились с двумя книгами, которые мы рекомендуем изучить и вам:

- *Арчер Т. Основы C#. Новейшие технологии/Пер. с англ. — М.: Русская Редакция, 2001. — 448 с.;*
- *Эпплман Д. Переход на VB .NET: стратегии, концепции, код/Пер. с англ. — Питер, 2002. — 464 с.*

Хотя мы профаны в C++/C#, в первой книге мы нашли много интересной и полезной информации. Что касается второй, то мы настоятельно рекомендуем прочитать ее всем, кто собирается заниматься программированием (и даже тем, кто уже от этого отказался). В книге очень хорошо рассказано о VB .NET и о .NET. Но самое замечательное в этом издании — общий подход к восприятию современных информационных технологий

На этом мы заканчиваем свои "Советы...". Но хотим еще раз предупредить: ***переход на VB .NET будет весьма непростым и потребует от вас серьезных усилий. Впрочем, выбора у вас просто нет...***

* * *

Оглавление

Совет 1. Используйте содержательные имена объектов 2
Совет 2. Оптимизируйте время загрузки нового проекта 4
Совет 3 (глобальный). Используйте функции Windows API 5
Совет 3a. Как реализовать в VB/Win функции MKx$/CVx 6
Совет 4. Используйте разнообразные источники информации по Visual Basic 9
Совет 5. Выполните "очистку" своего проекта перед созданием окончательного варианта EXE-модуля 12
Совет 6. Будьте внимательны при использовании оператора DIM 12
Совет 7. Сохраняйте свои программы в формате TEXT, а не BINARY 12
Совет 8. Читайте литературу, русскую и американскую 13
Совет 9. Используйте оператор Option Explicit 14
Совет 10. Следите за правильным использованием типов данных 15
Совет 11. Используйте оптимальные программные конструкции 15
Совет 12. Пишите комментарии в программе 17
Совет 13. Копируйте структуру меню в разные формы 17
Совет 13a. Как выделить текст при попадании в поле 17
Совет 13b. Как реализовать режим замены в текстовом окне 17
Совет 13c. Как сделать, чтобы клавиша Tab обрабатывалась как обычный символ 18
Совет 14. Выделение нескольких имен файлов в диалоговом окне Open File 18
Совет 15. Закрывайте другую программу из своего приложения 19
Совет 16. Удаляйте код после удаления соответствующего элемента управления 19
Совет 17. Производите очистку данных при выгрузке форм 19
Совет 18. Реагируйте на появление ошибок GPF правильно 19
Совет 19. Изменить последовательность прохождения элементов управления совсем не трудно 20
Совет 20. Избегайте длинных имен элементов управления и форм 20
Совет 21. Используйте Esc для выхода из программы-справки 20
Совет 22. Иногда полезно редактировать MAK-файл вручную 21
Совет 23. Разбивайте большие приложения на части 21
Совет 24. Используйте префикс в именах переменных 21
Совет 25. Используйте программный код для прекращения работы Windows 95 и даже для перезапуска Windows 22
Совет 26. Используйте клавиши управления курсором для настройки расположения или размеров элементов управления 23
Совет 27. Создавайте дополнительные средства (add-ins) 23
Совет 28. Как определить, закончено ли выполнение 32-разрядной программы, запущенной из вашего приложения 24

Совет 29. Как работать с наборами элементов (collections) 24
Совет 30. Оптимизируйте процедуры установки значений свойств 24
Совет 31. Следите за тем, что вы передаете в OLE-сервера 26
Совет 32. Снабжайте документами свои OLE-серверы 26
Совет 33. Что делать с ошибками OLE-сервера 26
Совет 34. Как обрабатывать объекты OLE-сервера 26
Совет 35. Будьте осторожны, когда завершаете работу OLE-серверов 26
Совет 36. Установите все переменные объекта равными Nothing (Пусто) 27
Совет 37. Используйте самый конкретный тип объектов из всех имеющихся 27
Совет 38. Используйте подстановку объектов 27
Совет 39. Тестируйте различные варианты реализации своих программ с целью выбора оптимального 27
Совет 40. Минимизируйте количество повторяемых перекодировок OLE 30
Совет 41. Используйте in-process серверы при любой возможности 30
Совет 42. Как ускорить вызовы OLE-сервера 30
Совет 43. Инициализация DLL-библиотеки, представленной в виде in-process OLE-сервера 30
Совет 44. Тестирование DLL-библиотеки, представленной в виде in-process OLE-сервера 30
Совет 45. Тестируйте in-process OLE-сервер как out-of-process сервер 31
Совет 46. Как выгрузить сервер из памяти 31
Совет 47. Как установить MousePointer в in-process OLE-сервер 31
Совет 48. Как передать массив элементов управления 31
VB 5.0 — хорошие новости для России 31
Совет 49. Устанавливайте текст в элемент управления Masked Edit программным образом 33
Совет 50. Традиционный: используйте самые быстрые конструкции 33
Совет 51. Модифицируйте (если это нужно) режим Tab при вводе данных 33
Совет 52. Простой способ переключения флагов 34
Совет 53. Используйте статические переменные (когда нужно) 35
Совет 54. Будьте внимательны при работе с динамическими массивами 36
Совет 55. Будьте внимательны при переходе из Win16 в Win32 39
Совет 56. Как определить, с каким приложением вы работаете в VBA 41
Совет 57. Используйте функцию Len для проверки пустой строковой переменной 42
Совет 58. Как сделать невидимыми на форме все ее элементы управления 42
Совет 59. Быстрый поиск строки в Combo Box 42
Совет 60. Как выгрузить все формы 43
Совет 61. Используйте для ожидания функцию API Sleep 43
Совет 62. Программное определение режимов "Run Time" и "Design Mode" 43
Совет 63. Как загрузить форму VB 4.0 в среде VB 3.0 43
Совет 64. Проверка существования файла 44
Совет 65. Создание многоуровневых каталогов 44

Совет 66. Редактирование ячеек Grid 45
Совет 67. Как сделать Beep в VB4 45
Совет 68. Простой способ очистки выбранных элементов списка 45
Совет 69. Перемещение элементов списка 45
Совет 70. Как посмотреть методы и свойства объекта 46
Совет 71. Определение типа элемента управления 46
Совет 72. Как преобразовать приложения для Access 2.0 в приложения для Access 7.0 47
Совет 73. Используйте параметр Optional при вызове процедур 48
Совет 74. Как узнать разрешающую способность монитора 48
Совет 75. Анализ имени файла 49
Совет 76 . Как зарегистрировать OCX-элементы и OLE-сервера 50
Совет 77. Совместное использование массивов для нескольких форм 51
Совет 78. Как создать быстрые клавиши в Win95 51
Совет 79. Как обойти все элементы дерева (TreeView Control) 52
Совет 80. Будьте внимательны при работе с константами типа Long 53
Совет 81. Как обеспечить числовой ввод 55
Совет 82. Как выполнить доступ к защищенным базам данных из VB4 55
Совет 83. Используйте некоторые правила написания кода в VB3 для его простого переноса в VB4 57
Совет 84. Как включить 16-разрядную версию Crystal OCX в прикладную программу 57
Совет 85. Как сохранить текущее значение индекса списка 58
Совет 86. Как уменьшить размеры приложения 58
Совет 87. Будьте внимательны при обращениях к функциям API 60
Совет 87a. Как проиграть Wav-файл 61
Совет 88. Будьте внимательны при установке VB 5.0. 62
Совет 88a. Индексирование русскоязычных баз данных 63
Совет 89. Управляйте режимами преобразования, сортировки и сравнения символьных переменных. 64
Совет 90. Простой алгоритм преобразования прописных букв в строчные и наоборот 65
Совет 91. Установка специальных шрифтов приложения 66
Совет 92. Автоматическая установка кавычек в строковых переменных 67
Совет 93. Используйте метод Refresh при работе с элементом управления SpinButton 68
Совет 94. Обращение к процедуре формы 68
Совет 95. Используйте свойство StartMode объекта App 69
Совет 96. Используйте коллекцию Properties для объектов доступа к данным 69
Совет 97. Добавьте свойства к объектам доступа к данным 69
Совет 98. Нельзя создавать библиотеку OLE DLL, когда класс Public установлен равным Creatable — Single Use 69
Совет 99. Помните об учете регистра в названиях DLL функций 70
Совет 100. Используйте согласованное превращение букв в идентификаторах VB 70

Совет 101. Создание двух версий одного проекта 70
Совет 102. Опробуйте нижеследующие рекомендации по компиляции для защиты и оптимизации ваших программ 71
Совет 103. Любителям программирования в кодах — вы можете читать P-код 71
Совет 104. Проблемы с бета-версией VB 5.0 72
Совет 105. При отладке не забывайте о сохранении проекта 72
Совет 106. Контролируйте установленные на компьютере шрифты 73
Совет 107. Новая утилита регистрации элементов управления 74
Совет 108. Управление шириной заголовков столбцов 74
Совет 109. Используйте короткие имена файлов (VB4 32, VB5) 75
Совет 110. Как обрезать полное имя файла до нужной длины (любые версии Basic) 75
Совет 111. Используйте смысловые имена при работе с элементом управления DBGrid (VB4 16/32) 76
Совет 112. Структуры данных могут быть более сложными (VB4 16/32, VB5) 76
Совет 113. Быстрая проверка выходных дней (все версии Basic) 77
Совет 114. Проверка високосного года (все версии Basic) 77
Совет 115. Динамическое управление положением элементов управления (VB4 16/32, VB 5) 77
Совет 116. Используйте функции API для работы с файлами (все версии VB) 78
Совет 117. Работа с беззнаковыми целыми числами (все версии Basic) 80
Совет 118. Как определить длину файла (все версии VB) 81
Совет 119. Используйте функции API для перекодировки текста. 81
Совет 120. Простые процедуры преобразования символьных данных. 82
Совет 121. Будьте внимательны при преобразовании исходных текстов программ из DOS в Windows, и наоборот. 85
Совет 122. Используйте метод GetRows для сохранения полей базы данных в переменных 86
Совет 123. Вывод длинных элементов списка в виде подсказки ToolTip 87
Совет 125. Простая проверка ввода данных 88
Совет 126. Управление длиной элемента списка ComboBox 89
Совет 127. Выводите осмысленные сообщения об ошибках при вызовах функций Win32 API. 90
Совет 128. Увеличение и уменьшение даты с помощью клавиш [+] и [-] 91
Совет 129. Различайте термины KeyCode и KeyAscii 92
Совет 130. Если хотите, то создавайте окна произвольной формы 93
Совет 131. Управление размещением на экране элементов среды в VB5 94
Совет 132. Не используйте звуковой сигнал при нажатии клавиши Enter 94
Совет 133. Управление длиной элемента списка ComboBox, вариант 2 94
Совет 134. Передача адреса процедуры в функцию API с помощью AddressOf 97
Совет 135. Как ускорить создание строковых буферов 98
Совет 136. Управление вводом в текстовых полях 100
Совет 137. Как перетащить элементы из одного списка в другой 101

Совет 138. Используйте ключевое слово ParamArray для передачи произвольного числа параметров 101
Совет 139. Создание нового контекстного меню 102
Совет 140. Используйте функцию FreeFile для предотвращения конфликтов при открытии файлов 103
Совет 141. Как предупредить пользователей о разрешении экрана 103
Совет 142. Простой способ определения логической переменной 104
Совет 143. Как быстро выделить текст для события GotFocus 104
Совет 144a. Как принудительно изменить атрибут файла 105
Совет 144. Для тех, кто занимается геометрическими расчетами 107
Совет 145. Определение положения указателя линейки прокрутки 109
Совет 146. Создание числового счетчика 110
Совет 147. Как измерить протяженность текста 110
Совет 148. Используйте пользовательские диалоговые окна при редактировании ячеек элемента управления DBGrid 111
Совет 149. При формировании констант типа Long используйте суффикс & 111
Совет 150. Использование параметра Alias при работе с функциями API 112
Совет 151. Как дать возможность пользователям отменить выгрузку форм 113
Совет 152. Создание "автоматического" поля текста 114
Совет 153. Управление кнопками Minimize/Maximize на MDI-форме 115
Совет 154. Как сделать видимыми кнопки Minimize/Maximize 116
Совет 155. Используйте VB API Viewer 116
Совет 156. Автоматическое выделение элементов списка 119
Совет 157. Применяйте повторно используемые процедуры 120
Листинг 1 120
Совет 158. Как очистить все текстовые окна 121
Совет 159. Используйте клавиатуру для управления размерами и расположением элементов управления 121
Совет 160. Эмуляция события Click для правой кнопки мыши 121
Совет 161. Установка курсора мыши в нужное место на форме 122
Совет 162. Используйте массивы элементов управления 123
Совет 163. Код, расположенный после вызова события Unload Me, препятствует закрытию формы 123
Совет 164. Как прочитать имена стандартных каталогов 124
Листинг 2 125
Совет 165. Определитесь - что такое текущий и временный каталоги? 125
Совет 166. Включение/выключение всех элементов управления в массиве 127
Совет 167. Сортировка пронумерованных элементов в окне списка 127
Листинг 3 128
Совет 168. Убедитесь, что вы закрыли все объекты доступа к данным при выходе из программы 129

Совет 169. Проверка наличия файла из любого места программы 129
Совет 170. Простой способ генерации случайных чисел 130
Совет 171. Используйте функцию CopyMemory из Win32 API 131
Совет 172. Копирование областей памяти в DOS 132
Совет 173. Как реализовать функции MKx$/CVx в VB/Win 133
Совет 174. Как сделать фон формы в виде палитры цветов 134
Совет 175. Помните о свойстве KeyPreview для формы 135
Совет 176. Использование экзотических "быстрых" клавиш в меню 135
Совет 177. Используйте свойства Default и Cancel для командных кнопок 136
Совет 178. Как определить имя накопителя CD-ROM 136
Совет 179. Как избежать ненужного обновления наборов записей 137
Совет 180. Упрощайте программный код 137
Совет 181. Будьте внимательны при работе с повторно используемыми BAS-компонентами в VB и VBA 137
Совет 182. Реализация функции "ожидания" в VB 142
Совет 183. Поддержка нумерации версий ваших VB-программ 142
Совет 184. Создание собственного хранителя экрана 143
Совет 185. Используйте WithEvents для добавления новых функций к элементу управления 143
Совет 186. Изменение набора изображений в элементе управления ImageList, связанном с элементом управления Toolbar 145
Совет 187. Загрузка элементов управления ActiveX 145
Совет 188. Использование типа Date с источником данных ADO 147
Совет 190. Экспорт содержимого элемента управления Grid в текстовый файл 148
Совет 191. Как определять високосный год 149
Совет 192. Форматирование числа при выводе 150
Совет 193. Как измерять временные интервалы 150
Совет 194. Как узнать значения кодовых таблиц 151
Совет 195. Как запустить Web-браузер из VB-приложения 152
Совет 196. Чтобы избежать появления ошибок, преобразовывайте значения NULL в пустые строковые переменные 152
Совет 197. Создание Word-документов при помощи VB-кода 153
Совет 198. Разбор полей строковой переменной при помощи функции SPLIT 153
Совет 199. Создайте свою собственную функцию Format 154
Совет 200. Как осуществить выход из Windows с помощью VB 154
Совет 201. Для ускорения процесса загрузки изменяйте базовые адреса компонентов типа in-process 155
Совет 202. Зачем нужен параметр Alias в операторе Declare 155
Совет 203. Помните: VB использует кодировку ANSI при обращении к API-функциям. 156
Совет 204. Как нарисовать рамку на форме без помощи элемента управления Frame 157

Совет 205. Как организовать просмотр каталогов 158
Совет 206. Удаление всех выделенных элементов в списке 159
Совет 207. Как сделать горизонтальную линейку прокрутки в элементе управления RichTextBox 159
Совет 208. Добавление новой строки к тексту в элементе управления TextBox 160
Совет 209. Как заставить VB 6.0 открывать окно кода в "развернутом" режиме 160
Совет 210. Специальные символы для объявления типа переменной 161
Совет 211. Создание объекта Excel в VB 161
Совет 212. Пусть функция DateDiff разбирается с датами 161
Совет 213. Для аварийного прерывания программы используйте Ctrl+Pause 162
Совет 214. Как увидеть начало выделенного текста 162
Совет 215. Как прочитать серийный номер диска 163
Совет 216. Применение элемента управления Label в качестве разделителя 164
Совет 217a. Как передавать данные между формами 165
Совет 217. Использование функции Split для вычисления количества подстрок 168
Совет 218. Будьте осторожны при использовании методов Delete объекта FileSystemObject в VB 169
Совет 219. Заменяйте длинные строки, содержащие путь, на объектные переменные (продолжение Совета 218) 169
Совет 220. Для ограничения размеров VB-формы пользуйтесь элементом управления ActiveX 169
Совет 221. Как показать стандартное диалоговое окно Properties для файла 170
Совет 222. Настройка цветов и шрифтов в строке состояния приложения 171
Совет 223. Как выровнять текст в командной кнопке 172
Совет 224. Управление шириной столбцов элемента управления DBGrid в процессе выполнения программы 173
Совет 225. Простой способ запомнить события запуска VB-формы 174
Совет 226. Обмен переменными между разными приложениями 174
Совет 227. Представление числового значения прописью 175
Листинг 277. Функция SummaString выполняет задачу "запись числи прописью" 177
Совет 228. Как практически воспользоваться функцией SummaString? 179
Листинг 2. Вспомогательный вариант "сумма прописью" 180
Совет 229. Используйте реентерабельные и рекурсивные процедуры. Но осторожно! 181
Совет 230. Используйте рекурсию для передвижения по файлам 184
Листинг 230. Перемещение по дереву файловой системы 185
Совет 231. Преобразование имен файлов в определенный регистр 186
Совет 232. Используйте Preserve для сохранения содержимого массива 187
Совет 233. Используйте FreeFile для получения номера файла. Но будьте при этом бдительны! 188
Совет 234. Используйте свойство ItemData элемента управления ListBox для хранения идентификаторов 188

Совет 235. Как правильно сгенерировать значения ADO RecordCount в VB 189
Совет 236. Пусть VB сам определяет, есть ли в дисководе CD-ROM компакт-диск или нет 190
Совет 237. Как добавить новый элемент к системному меню формы 190
Совет 238. Как быстро осуществить проверку на наличие определенного элемента в списке 191
Совет 239. Используйте для объектов ADO родные драйверы OLEDB вместо драйверов ODBC 191
Совет 240. Как модифицировать элемент управления DataGrid при помощи изменения набора записей 192
Совет 241. Как запустить на выполнение VB-приложение в процессе инициализации Windows 192
Совет 242. Как восстановить короткое имя файла в VB без помощи API-функций 193
Совет 243. Создавайте временные VB-файлы с помощью API-функции 193
Совет 244. Как преобразовать число в строку с фиксированным количеством цифр 194
Совет 245. Используйте режим Option Explicit 194
Совет 246. Как сделать временную задержку 196
Совет 247. Как сделать заставку при старте VB-программы или VBA-документа 197
Совет 248. Как изучать программирование в среде VBA 200
Совет 249. Как обрабатывать строки на VBA и каковы преимущества для этого в MS Office 2000 201
Совет 250. Как перекодировать почтовые сообщения 201
Совет 251. Как копировать ячейки между рабочими книгами 202
Совет 252. Как прервать вычислительный процесс 204
Совет 253. Как решить проблему с сохранением проектов с цифровой подписью 205
Совет 254. Как автоматически определить кодовую таблицу для русских текстов 206
Листинг 254. Процедуры CodeTableTest и NumberTableTest для совета 254 208
Совет 255. Как определить кодировку текста: еще один вариант 209
Листинг 255. Код утилиты RusCode2 для Совета 255 210
Совет 256. Программное обращение к CommonDialog 211
Листинг 256. Обращение к диалоговому окну "Открыть/Сохранить" через WinAPI 212
Совет 257. Как узнать, существует ли файл? 214
Совет 258. Как выбрать имя каталога 214
Совет 259. Какие операторы лучше использовать: Print # или Write #? 215
Совет 260. Где хранить INI-файл 217
Совет 261. Как обеспечить совместимость VBA и VB 219
Совет 262. Как контролировать макросы в шаблонах 220
Совет 263. Использование даты в SQL-запросах 220
Совет 264. Программный сброс хранителя экрана 221
Листинг 264. Имитация нажатия клавиши клавиатуры 222
Совет 265. Как подключить VB-процедуры обратного вызова к API-функции 223

Совет 266. Как избежать ошибок сравнения битовой маскировки при использовании API-функций в VB 224
Совет 267. Используйте VB-объект RegExp для проверки синтаксиса адреса электронной почты 225
Совет 268. Как осуществить прокрутку элемента управления ListView в VB до заданного элемента 225
Совет 269. Работа в VB с формулами, представленными в виде строковых переменных 226
Совет 270. Как сделать невидимым курсор мыши в VB-приложении 226
Новость: создан Web-архив повторно используемого кода 227
Нам пишут, мы отвечаем 228
Глобальный совет: работайте с лицензионными копиями программ 230
Совет 271. Избегайте использования переменных типа Variant 231
Совет 272. Вывод анимационных GIF-файлов в VB 233
Совет 273. Как прочитать значения Системного Реестра в VB без помощи функций API 233
Совет 274. Реализация проверки правописания в RTF-документах в VB 234
Совет 275. Как реализовать ссылки к набору записей объекта Command, содержащегося в конструкторе Data Environment 234
Совет 276. Сжимайте длинные имена файлов с помощью библиотеки SHLWAPI 234
Совет 277. Еще один способ чтения нескольких элементов списка 235
Совет 278. Как можно просто связать элементы двух списков 236
Совет 279. Как реализовать работу с несколькими дисководами CD-ROM с помощью MCI-интерфейса 237
Совет 280. Как обрабатывать события, связанные с изменением свойств шрифта 238
Совет 281. Как определить, какой именно объект не может быть создан 238
Совет 282. Используйте возможности Office для проверки правописания в элементе управления RichTextBox 239
Совет 283. Используйте библиотеку ADOX, чтобы установить существование внутренних объектов баз данных 240
Совет 284. Используйте оператор SendKeys 240
Совет 285. Как получить значения цветовой палитры 241
Совет 286. Будьте внимательны при использовании оператора Debug.Print 242
Совет 287. Как упростить математические вычисления в VBA 243
Совет 288. Как вычислить интервал между двумя датами, измеряемый в разных единицах 244
Совет 289. Как правильно прочитать имя каталога 246
Совет 290. Как заполнить поля электронного сообщения 247
Совет 291. Как открыть VB-файл в Notepad или в любом другом текстовом редакторе 249
Совет 292. Как ускорить операцию деления чисел с плавающей запятой 250
Совет 293. Как ускорить выполнение операций ввода-вывода для файлов 250
Совет 294. Как создать дополнение, закрывающее все окна проекта 251
Совет 295. Используйте ключевое слово WithEvents для связи форм MDI и MDIChild 252
Совет 296. Помните об операторе Option Base 253

Совет 297. Экспорт таблиц в виде текстовых файлов 254
Совет 298. Быстрый своппинг строковых переменных 255
Совет 299. Динамическое управление меню 256
Совет 300. Используйте каталог Template 257
Совет 301. Используйте Microsoft Forms 2.0 Form (когда это нужно) 257
Совет 302. Используйте элемент управления IE Timer 258
Совет 303. Как погрузить приложение в "глубокий сон" 259
Совет 304. Управление расстоянием между столбцами в ListBox 259
Совет 305. Где хранятся настройки панелей инструментов MS Office 261
Совет 306. Управление порядком перемещения по элементам управления 263
Совет 307. Как удалить каталог 266
Совет 308. Как удалить непустой каталог 267
Совет 309. Как получить сводку о файле MP3 268
Совет 310. Приоритетность использования оператора With 269
Совет 311. Следите за состоянием батарей ноутбука 270
Совет 312. Следите за правильным использованием типов данных 270
Совет 313. Как контролировать наличие кавычек в названиях 271
Совет 314. Вывод подсказок на панель StatusBar 273
Совет 315. Программное управление раскладкой клавиатуры 273
Совет 316. Простые алгоритмы шифрования данных 274
Совет 317. Как вывести информацию в элементе управления WebBrowser, не пользуясь HTML-файлом 275
Совет 318. Как преобразовать OLE_COLOR в фактическое значение функции RGB 276
Совет 319. Один из способов создания уникального строкового идентификатора 277
Совет 320. Как с помощью метода OpenSchema получить информацию о структуре базы данных 278
Совет 321. Как обнаружить неиспользуемые объекты 279
Совет 322. Получение даты последней коррекции файла 280
Совет 323. Преобразование кода цвета из DOS в Windows 281
Совет 324. Как отличить обычные элементы управления от компонентов массива 282
Совет 325. Преобразование числа в двоичное представление и наоборот 283
Совет 326. Функции с двухсторонним обращением 284
Совет 327. Преобразование величины размера памяти 285
Совет 328. Переключение вида заголовков элемента управления ListView 286
Совет 329. Генерация динамических HTML-страниц 286
Совет 330. Как отлаживать ASP-скрипты 288
Совет 331. Как улучшить читаемость кода 289
Совет 332. Проверка на недействительные символы 291
Совет 333. Проверка дубликатности элементов списка 292
Совет 334. Как организовать выбор каталога 292

Совет 335. Как передать параметры между процедурами 292
Совет 336. Формирование сокращенного имени каталога 293
Совет 337. Инициализация ActiveX DLL 294
Совет 338. Как определить события инициализации и завершения работы ActiveX DLL....294
Совет 339. Как определить полное имя Host-приложения 295
Совет 340. Как ввести табуляцию в Textbox 295
Совет 341. Динамическая передача данных из списка в текстовое поле 296
Совет 342. Как переписать удаленные файлы в Recycle Bin 296
Совет 343. Поддержка контекстного меню для Rich Textbox 297
Совет 344. Дайте возможность пользователям изменять размеры элементов управления ...298
Совет 345. В циклах лучше используйте GetInputState 299
Совет 346. Работа с Интернетом в VB-приложениях....300
Совет 347. Как определить текущие размеры экрана....300
Совет 348. Как установить связь с Microsoft Excel с помощью OLE DB....301
Совет 349. Простой ввод данных в элементе управления MSFlexGrid 302
Совет 350. Десять советов "как не нужно поступать при работе с ASP" 303
Совет 351. Пишите замечания, присылайте советы, задавайте вопросы 304
Совет 352. Как решить проблему с VisData 304
Совет 353. Внимание при работе с булевыми типами данных 305
Совет 354. Как обеспечить совместимость между VBA- и VB-проектами 306
Совет 355. Используйте свойство Button для элемента управления DataGrid 307
Совет 356. Как сделать Help для своего приложения 307
Совет 357. Преобразование текстового файла в набор данных ADO 307
Совет 358. Как расширить массив элементов управления во время выполнения....308
Совет 359. Работа с реестром и INI-файлами с помощью System.PrivateProfileString....308
Совет 360. Вместо DoEvents отслеживайте реальные события....310
Совет 361. Избегайте неявного преобразования типов данных....311
Совет 362. Как узнать список папок Outlook 312
Совет 363. Как добавить новый контакт в папку Outlook....314
Совет 364. Как обрабатывать входящие письма 315
Совет 365. Как вставить текст в создаваемое письмо 315
Совет 366. Как узнать адрес отправителя письма в Outlook 2000 316
Совет 367. Поиск файлов с помощью объекта FileSearch 316
Совет 368. Сортировка содержимого ListView....318
Совет 369. Как определить имя дисковода CD-ROM с помощью FileSystemObject....319
Совет 370. Как передать текст из Rich Textbox в Microsoft Word 319
Совет 371. Как установить VBA SDK....320
Совет 372. Как добавить иконку к меню или к кнопке панели инструментов Office 320
Совет 373. Как определить ID элементов меню и панелей инструментов 321

Совет 374. Как записать иконку для меню Add-Ins в среде VB 322
Совет 375. Как избавиться от ненужных окон-сообщений 323
Совет 376. Используйте XML Parser 324
Совет 377. Используйте новшества MSXML 3.0 325
Совет 378. Установка кодировки в MS XML Parser 3.0 326
Совет 379. Преобразование последовательности байтов в строку шестнадцатеричных символов 326
Совет 380. Используйте автоматически запускаемые макросы Word 328
Совет 381. Создание строки меню 328
Совет 382. Как узнать значение свойств документа Word 330
Совет 383. Копирование набора данных в XLS-файл с помощью метода CopyFromRecordset 331
Совет 384. Использование Script Control в VBA 332
Совет 385. На что нужно обратить внимание при замере интервалов времени 333
Совет 386. Не применяйте неявного преобразования данных 334
Совет 387. Передача XML-данных в виде атрибутов 336
Листинг 387. Вывод XML-данных в виде элементов и атрибутов 337
Совет 388. Передача адреса процедуры через структуру 338
Совет 389. Программная регистрация ActiveX DLL и OCX 339
Совет 390. Быстрое перемещение между процедурами 340
Совет 391. Как посчитать число строк в текстовом окне 340
Совет 392. Как определить позицию курсора в Rich Textbox 340
Совет 393. Программная имитация нажатия кнопок 341
Совет 394. Динамическое управление заголовками колонок DataGrid 341
Совет 395. А вы знаете о свойстве LockControls формы? 342
Совет 396. Используйте функцию ExtFloodFill для цветной заливки поверхности 342
Листинг 396-1. Модуль FloodFll.bas — процедура Paint для заливки фигуры произвольных очертаний 345
Листинг 396-2. Модуль Flood.frm — демонстрационный пример практического использования процедуры Paint 346
Листинг 396-3. Модуль AddProc.bas — вспомогательные процедуры для формы Flood.frm ..347
Совет 397. Как использовать свойство Filter при работе с ADO 349
Совет 398. В который раз повторяем: используйте режим Option Explicit 349
Совет 399. Задавайте вопросы в понятной форме 350
Совет 400. Используйте режим раннего связывания 350
Совет 401. Управляйте режимами программного контроля ошибок 351
Совет 402. Быстрый поиск по списку 352
Совет 403. При программном создании наборов данных указывайте размеры полей 353
Совет 404. Программное отключение предупреждающих сообщений в приложениях MS Office 354
Совет 405. Как выводить на экран формы в динамическом режиме 354

Совет 406. Как узнать параметры диска 355
Совет 407. Как создать дубликатный набор данных 355
Совет 408. Как узнать размер свободного места на диске 356
Совет 409. Блокировка Windows 2000 356
Совет 410. Настройка интерфейса VB/VBA 357
Совет 411. Добавьте команды управления комментариями к среде разработки 358
Совет 412. Как прочитать адреса отправленных писем 359
Совет 413. Использование пробелов в именах таблиц баз данных ADO 359
Совет 414. Как передать Null в API-функцию 359
Совет 415. Как вставить колонки в VB MSFlexGrid 361
Совет 416. Динамическое увеличение динамического массива 361
Совет 417. Как работать с адресной книгой MS Exchange 362
Совет 418. Как отключить ADO Recordset, созданный с помощью объекта Command 363
Совет 419. Как использовать символ @ в именах параметров SQL Server 363
Совет 420. Генерация строк соединения для OLE DB 364
Совет 421. Как создавать ISAM-файлы 364
Совет 422. Защита от макросов при использовании глобальных шаблонов 365
Совет 423. Управление подключением макросов в приложениях Office 366
Совет 424. Использование цифровой подписи в Office 2000/XP 368
Совет 425. Идентификация элементов управления 374
Совет 426. Предотвращение многократного запуска VB-программы на одном компьютере 375
Совет 427. Удостоверьтесь, что используете нужный провайдер при создании структурированных наборов данных ADO 375
Совет 428. Программная реализация копирования экрана 375
Совет 429. Как открыть ниспадающее меню программным образом 376
Совет 430. Формирование кода цвета для HTML 377
Совет 431. Модернизация объекта Err 377
Совет 432. Как позволить пользователям держать командную кнопку в нажатом состоянии 378
Совет 433. Преобразование текстовых записей в массив с помощью Split 379
Совет 434. Как выгрузить VB-форму нажатием Esc 379
Совет 435. Как определить относительный адрес файла 380
Совет 436. Использование системных значков в стиле MsgBox 380
Совет 437. Как получить описание файла 381
Совет 438. Создание MDI-формы без строки Caption 381
Совет 439. Как преобразовать цветное изображение в черно-белое (Grayscale) 382
Совет 440. Как ограничить ввод для ComboBox 383
Совет 441. Копирование содержимого ListView в буфер обмена 383
Совет 442. Центровка формы с учетом системных панелей 384
Совет 443. Программная имитация щелчка мышью 384

Совет 444. Выполнение операций с помощью механизма Scripting 385
Заключительные советы тем, кто программирует на VB & VBA 386
"Сериал" длиною в шесть лет 386
Совет 445. Изучайте VB в контексте его исторического развития 387
Совет 446. Не спешите расставаться с VB 6.0 389
Совет 447. Помните о наших советах 389
Совет 448. Готовьтесь к VB .NET 391

* * *