Бесплатная электронная книга

УЧУСЬ VBA

Free unaffiliated eBook created from
Stack Overflow contributors.

#vba

1
1: VBA 2
2
2
Examples 2
Visual Basic Microsoft Office 2
Hello World 6
6
6
7
7
7
2: API- 9
9
9
Examples 11
API 11
Windows API - (1 2) 13
Windows API - (2 2) 18
Mac API 21
23
FTP API 23
3: CreateObject vs. GetObject 27
27
Examples 27
GetObject CreateObject 27
4: Scripting.FileSystemObject 29
Examples 29
FileSystemObject 29
FileSystemObject 29
FileSystemObject 30

FileSystemObject 30
FileSystemObject 30
..... 31
Strip 32
..... 33
..... 33
FSO.BuildPath 33
5: 34
..... 34
..... 34
..... 34
Examples 35
VBScript 35
..... 35
..... 36
..... 37
Internet Explorer 37
Internet Explorer Objec 37
- 38
..... 39
Microsoft HTML Object IE 40
IE 40
6: 41
..... 41
Examples 41
VB_Name 41
VB_GlobalNameSpace 41
VB_Createable 42
VB_PredeclaredId 42
..... 42
..... 42
VB_Exposed 42

VB_Description 43
VB_ [] UserMemId 43
44
For Each loop 44
7: 46
46
Examples 46
Len 46
LenB 46
` Len (myString) = 0 ` over `If myString =" "Then` 46
8: 48
48
Examples 48
- 48
- 49
9: Option VBA 52
52
52
53
Examples 53
53
{Binary | | } 54
54
54
55
{0 | 1} 55
0: 56
1 56
1: 56
10: 58
58

58

Examples....59

....59

....60

....61

....62

....64

....64

....64

....65

11: 66

....66

Examples....66

....66

REM....67

12: 68

....68

Examples....68

&....68

Join....68

13: , ...69

Examples....69

....69

....70

, ...70

....70

....**71**

....71

....72

....72

14: VBA- / - 74

Examples....74

SELFCERT.EXE 74
15: 88
Examples 88
..... 88
..... 88
..... 89
DateTime 89
..... 89
IsDate () 90
..... 90
DatePart () 92
..... 93
DateDiff () 93
DateAdd () 93
..... 94
CDate () 94
DateSerial () 95
16: 97
Examples 97
VBA 97
..... 97
..... 97
..... 97
..... 97
Split 98
..... 100
... . 100
... . 100
( ) 101
..... 101
..... 101
..... 102

102
( ) 103
103
103
103
106
106
106
109
17: 113
113
Examples 113
String, n 113
String Space n- 113
18: 114
114
Examples 114
VBA 114
115
19: 117
Examples 117
117
Error 118
119
119
120
. 121
122
123
20: Scripting.Dictionary 124
124

Examples....124
....124
Scripting.Dictionary (Maximum, Count)....126
Scripting.Dictionary....128
21: - VBA....130
Examples....130
....130
, ...130
....131
....131
....131
....133
....134
- 134
....136
22: 138
....138
Examples....138
....138
....138
....138
....138
Mid....139
....139
23: 141
Examples....141
....141
....141
....141
....142
....142
....144

144

145

(Const) 145

146

147

147

148

150

150

24: 153

153

Examples 153

153

154

155

156

\ 158

25: VBA 161

161

Examples 161

«3»: GoSub 161

161

? 161

161

? 161

162

«6»: 162

162

? 162

162

? 162

163

«9»: 163
163
?163
163
?163
163
«13»: 164
164
?164
164
?164
'91': 164
164
?165
165
?165
165
«20»: 166
166
?166
166
?166
167
26: ByRef ByVal 168
168
168
168
Examples 168
ByRef And ByVal 168
ByRef 169
169
170

ByVal 171
ByVal 171
..... 171
27: 173
..... 173
Examples 173
Left Left $, 3 173
$, 3 173
Mid Mid $ 173
Trim - 174
28: 175
..... 175
Examples 175
InStr, , ... 175
InStr, ... 175
InStrRev, ... 175
29: 177
Examples 177
..... 177
..... 177
..... 177
..... 178
QueryClose 178
, ... 179
..... 179
QueryClose 180
..... 180
30: 182
..... 182
Examples 182
CStr 182
..... 182

StrConv 182
..... 183
31: 184
..... 184
..... 184
..... 184
Examples 184
..... 184
..... 184
..... 185
. ? 185
..... 185
..... 186
..... 186
..... 186
32: ADO 188
..... 188
Examples 188
..... 188
..... 189
..... 190
..... 191
33: FileSystemObject 193
..... 193
Examples 193
..... 193
..... 194
34: 196
..... 196
..... 196
Examples 196

196

196

35: 198

198

198

Examples 198

198

? 198

199

200

201

, 201

201

36: 203

Examples 203

203

204

206

37: 208

208

Examples 208

208

209

, 210

38: 212

Examples 212

212

212

213

39: 215

215

Examples....215
- 215
Excel 216
40: - , ...218
....218
Examples....218
....218
....218
VBA....219
41: 221
....221
Examples....221
....221
....222
42: 224
Examples....224
....224
....225
....226
....226
....227
....227
43: 229
Examples....229
....229
....230
....230
....231
....231
....232
....232
....232

233
233
233
234
234
LongPtr 236
236
44: 238
Examples 238
238
Declare Imports, Office 239
45: 241
241
Examples 241
241
46: 2GB + VBA 243
243
243
MICROSOFT 243
MICROSOFT 244
244
Examples 244
, , «Random», 244
248
Hash 250
: 250
251
254

Около

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: vba

It is an unofficial and free VBA ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official VBA.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

глава 1: Начало работы с VBA

замечания

В этом разделе представлен обзор того, что такое vba, и почему разработчик может захотеть его использовать.

Следует также упомянуть любые крупные темы в vba и ссылки на связанные темы. Поскольку документация для vba является новой, вам может потребоваться создать начальные версии этих связанных тем.

Версии

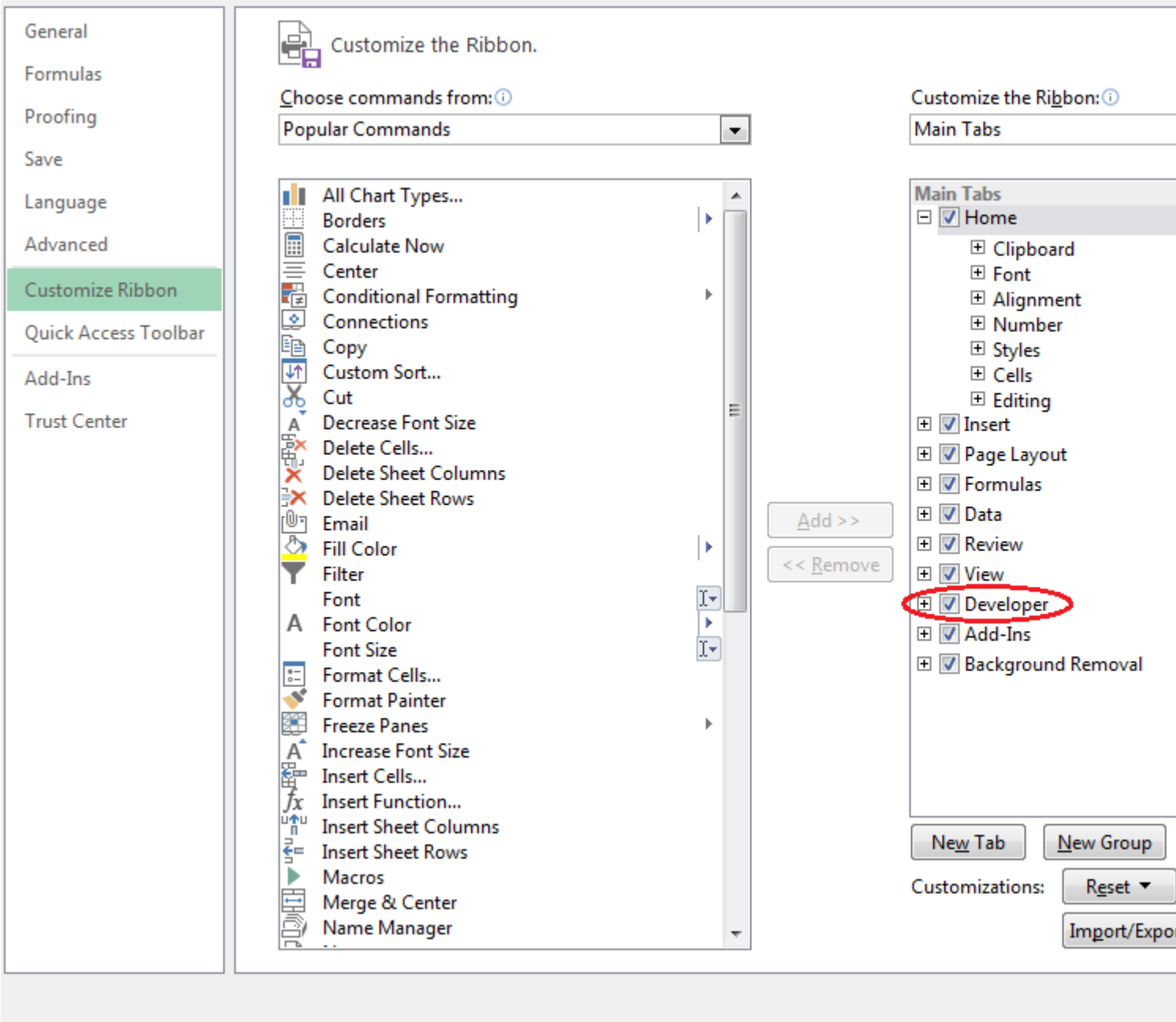
Версия	Офисные версии	Дата выпуска Примечания	Дата выхода
VBA6	? - 2007	[Когда-то после] [1]	1992-06-30
Vba7	2010 - 2016	[Blog.techkit.com] [2]	2010-04-15
VBA для Mac	2004, 2011 - 2016 гг.		2004-05-11

Examples

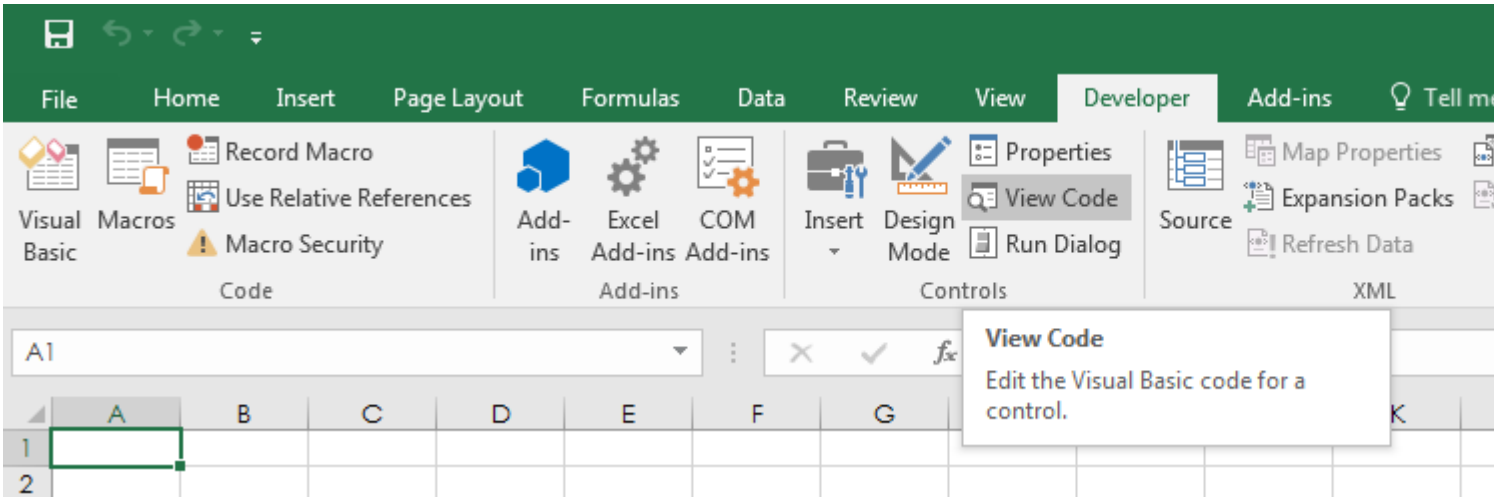
Доступ к редактору Visual Basic в Microsoft Office

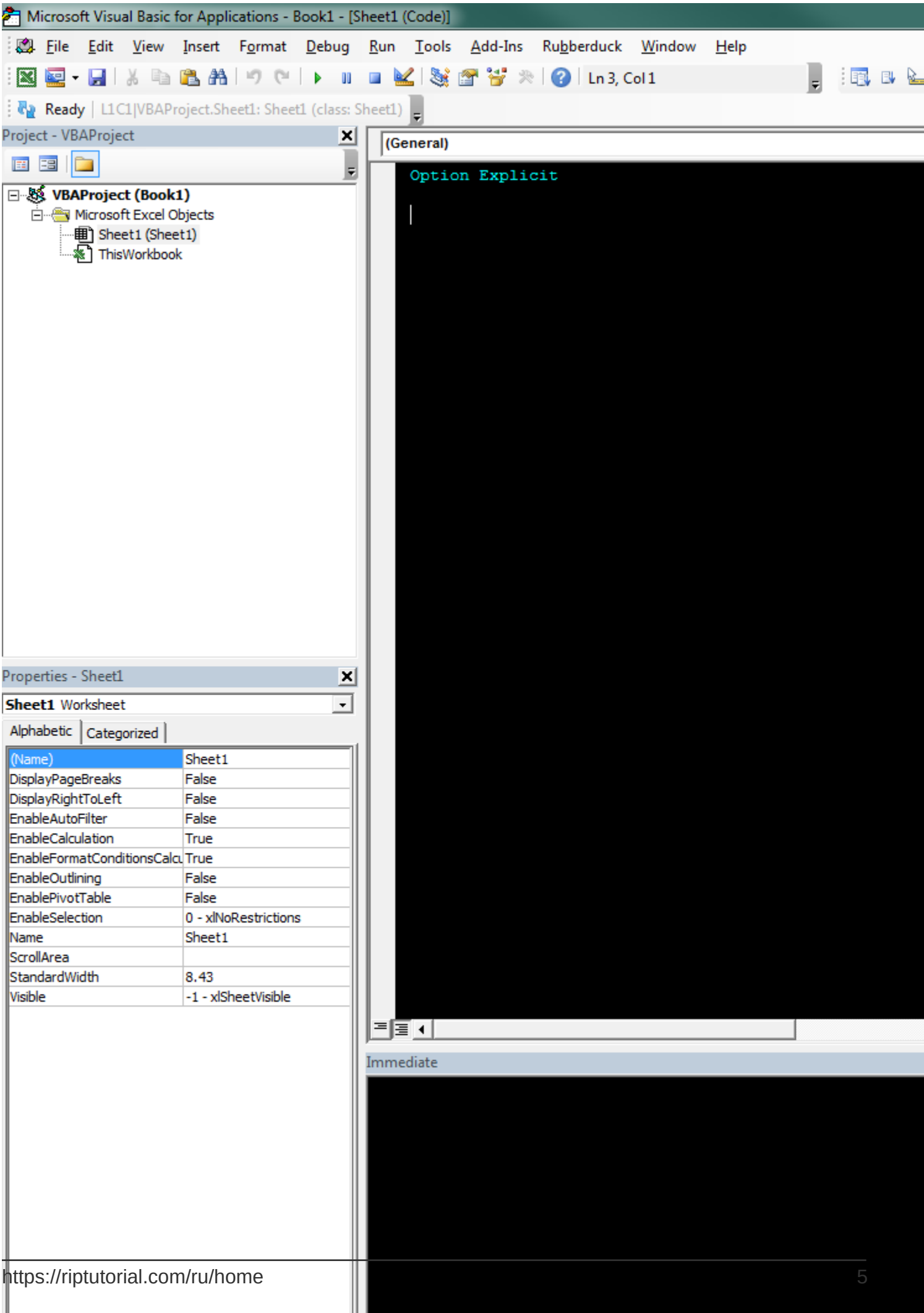
Вы можете открыть редактор VB в любом из приложений Microsoft Office, нажав `Alt` + `F11` или перейдите на вкладку «Разработчик» и нажав кнопку «Visual Basic». Если вы не видите вкладку «Разработчик» в ленте, проверьте, включено ли это.

По умолчанию вкладка «Разработчик» отключена. Чтобы включить вкладку «Разработчик», перейдите в «Файл -> Параметры», выберите «Настроить ленту» в списке слева. В правой части дерева «Настроить ленту» найдите элемент «Дерево разработчика» и установите флажок «Проверить флажок разработчика». Нажмите «ОК», чтобы закрыть диалоговое окно «Параметры».



Вкладка «Разработчик» теперь отображается в ленте, на которой вы можете щелкнуть « Visual Basic», чтобы открыть редактор Visual Basic. В качестве альтернативы вы можете нажать «Просмотреть код», чтобы непосредственно просмотреть панель кода активного элемента, например WorkSheet, Chart, Shape.


File
Home
Insert
Page Layout
Formulas
Data
Review
View
Developer
Add-ins
Visual Basic
Macros
Record Macro
Use Relative References
Macro Security
Code
Add-ins
Excel Add-ins
COM Add-ins
Add-ins
Insert
Design Mode
Properties
View Code
Run Dialog
Controls
Source
Map Properties
Expansion Packs
Refresh Data
XML
A1
View Code
Edit the Visual Basic code for a control.
A
B
C
D
E
F
G
1
2


Microsoft Visual Basic for Applications - Book1 - [Sheet1 (Code)]
File Edit View Insert Format Debug Run Tools Add-Ins Rubberduck Window Help
Ln 3, Col 1
Ready
L1C1|VBAProject.Sheet1: Sheet1 (class: Sheet1)
Project - VBAProject
VBAProject (Book1)
Microsoft Excel Objects
Sheet1 (Sheet1)
ThisWorkbook
(General)
Option Explicit
Properties - Sheet1
Sheet1 Worksheet
Alphabetic
Categorized
(Name) Sheet1
DisplayPageBreaks False
DisplayRightToLeft False
EnableAutoFilter False
EnableCalculation True
EnableFormatConditionsCalcu True
EnableOutlining False
EnablePivotTable False
EnableSelection 0 - xlNoRestrictions
Name Sheet1
ScrollArea
StandardWidth 8.43
Visible -1 - xlSheetVisible
Immediate

для автоматизации практически любых действий, которые могут выполняться в интерактивном режиме (вручную), а также предоставлять функциональные возможности, недоступные в Microsoft Office. VBA может создавать документ, добавлять в него текст, форматировать его, редактировать и сохранять, без вмешательства человека.

Первый модуль и Hello World

Чтобы начать кодирование в первую очередь, вы должны щелкнуть правой кнопкой мыши свой проект VBA в левом списке и добавить новый модуль. Ваш первый код *Hello-World* может выглядеть так:

```
Sub HelloWorld()
    MsgBox "Hello, World!"
End Sub
```

Чтобы протестировать его, нажмите *Play* -Button на панели инструментов или просто нажмите клавишу `F5` . Поздравляем! Вы создали свой собственный VBA-модуль.

отладка

Отладка - очень мощный способ более пристального взгляда и исправления некорректно работающего (или не работающего) кода.

Выполнить код шаг за шагом

Первое, что вам нужно сделать во время отладки, - это остановить код в определенных местах, а затем запустить его по очереди, чтобы узнать, произойдет ли то, что ожидается.

- Точка останова (`F9` , Debug - Toggle breakpoint): вы можете добавить точку останова в любую выполненную строку (например, не к объявлениям), когда выполнение достигает этой точки и останавливается и дает управление пользователю.
- Вы также можете добавить ключевое слово `Stop` в пустую строку, чтобы остановить код в этом месте во время выполнения. Это полезно, если, например, перед линиями объявлений, к которым вы не можете добавить точку останова с `F9`
- Шаг в (`F8` , Debug - Step in): выполняется только одна строка кода, если это вызов определенной пользователем функции / функции, то это выполняется по строкам.
- Step over (`Shift` + `F8` , Debug - Step over): выполняет одну строку кода, не вводит пользовательские подмножества / функции.
- Выйти (`Ctrl` + `Shift` + `F8` , Debug - выйти): выйти из текущего суб / функции (запустить код до конца).
- Запустите курсор (`Ctrl` + `F8` , Debug - Run to cursor): запустите код, пока не дойдете до строки с помощью курсора.
- Вы можете использовать `Debug.Print` для печати строк в окне Immediate во время

выполнения. Вы также можете использовать `Debug.?` как ярлык для `Debug.Print`

Окно часов

Запуск кода за строкой - это только первый шаг, нам нужно знать больше деталей, и один инструмент для этого - это окно часов (View - Watch), здесь вы можете увидеть значения определенных выражений. Чтобы добавить переменную в окно просмотра, выполните следующие действия:

- Щелкните его правой кнопкой мыши и выберите «Добавить часы».
- Щелкните правой кнопкой мыши в окне просмотра, выберите «Добавить часы».
- Перейдите в Debug - добавьте часы.

Когда вы добавляете новое выражение, вы можете выбрать, хотите ли вы просто увидеть его значение, или также нарушить выполнение кода, когда оно истинно или когда изменяется его значение.

Немедленное окно

Непосредственное окно позволяет выполнять произвольный код или печатать элементы, преследуя их либо ключевым словом `Print` либо одним вопросительным знаком « `?` ».

Некоторые примеры:

- `? ActiveSheet.Name` - возвращает имя активного листа
- `Print ActiveSheet.Name` - возвращает имя активного листа
- `? foo` - возвращает значение `foo` *
- `x = 10` наборов `x` до 10 *

* Получение / установка значений для переменных через окно Immediate может выполняться только во время выполнения

Отладка лучших практик

Всякий раз, когда ваш код не работает должным образом, первое, что вам нужно сделать, - это внимательно прочитать его, искать ошибки.

Если это не поможет, начните отладку; для коротких процедур может быть достаточно просто выполнить его по строкам, для более длинных вам, вероятно, необходимо установить точки останова или разрывы на наблюдаемые выражения, целью здесь является поиск строки, которая не работает должным образом.

Если у вас есть строка, которая дает неверный результат, но причина еще не ясна, попробуйте упростить выражения или заменить переменные константами, которые могут помочь понять, являются ли значения переменных неправильными.

Если вы все еще не можете его решить и обратитесь за помощью:

- Включите как можно большую часть своего кода для понимания вашей проблемы
- Если проблема не связана со значением переменных, замените их на константы. (так, вместо `Sheets(a*b*c+d^2).Range(addressOfRange)` написать `Sheets(4).Range("A2")`)
- Опишите, какая строка дает неправильное поведение и что это (ошибка, неправильный результат ...)

Прочитайте Начало работы с VBA онлайн: https://riptutorial.com/ru/vba/topic/802/начало-работы-с-vba

глава 2: API-запросы

Вступление

API означает интерфейс прикладного программирования

API для VBA подразумевает набор методов, которые позволяют прямое взаимодействие с операционной системой

Системные вызовы могут выполняться путем выполнения процедур, определенных в файлах DLL

замечания

Общие файлы библиотеки операционной среды (DLL):

Динамическая библиотека ссылок	Описание
Advapi32.dll	Библиотека дополнительных сервисов для API, включая многие вызовы безопасности и реестра
cOMDLG32.DLL	Общая библиотека API диалога
Gdi32.dll	Интерфейс API интерфейса графического интерфейса
Kernel32.dll	Поддержка 32-битного базового API ядра Windows
Lz32.dll	32-разрядные процедуры сжатия
Mpr.dll	Многоуровневая библиотека маршрутизаторов
Netapi32.dll	32-битная библиотека сетевого API
Shell32.dll	32-разрядная библиотека API оболочки
User32.dll	Библиотека для пользовательских интерфейсов
Version.dll	Библиотека версий
Winmm.dll	Мультимедийная библиотека Windows
WINSPOOL.DRV	Интерфейс диспетчера очереди печати, содержащий вызовы API очереди печати

Новые аргументы, используемые для системы 64:

Тип	Вещь	Описание
спецификатор	PtrSafe	Указывает, что оператор Declare совместим с 64-битными. Этот атрибут является обязательным для 64-битных систем
Тип данных	LongPtr	Переменный тип данных, который представляет собой 4-байтовый тип данных в 32-разрядных версиях и 8-байтовый тип данных в 64-разрядных версиях Office 2010. Это рекомендуемый способ объявления указателя или дескриптора для нового кода, но также для устаревшего кода, если он должен запускаться в 64-разрядной версии Office 2010. Он поддерживается только в среде исполнения VBA 7 на 32-разрядной и 64-разрядной версиях. Обратите внимание, что вы можете присвоить ему числовые значения, но не числовые типы
Тип данных	Долго долго	Это 8-байтовый тип данных, доступный только в 64-разрядных версиях Office 2010. Вы можете назначать числовые значения, но не числовые типы (чтобы избежать усечения)
преобразование	оператор	CLngPtr Преобразует простое выражение в тип данных LongPtr
преобразование	оператор	CLngLng Преобразует простое выражение в тип данных LongLong
функция	VarPtr	Конвертер вариантов. Возвращает LongPtr в 64-разрядных версиях, а длинный 32-разрядный (4 байта)
функция	ObjPtr	Конвертер объектов. Возвращает LongPtr в 64-разрядных версиях, а длинный 32-разрядный (4 байта)
функция	StrPtr	Преобразователь строк. Возвращает LongPtr в 64-разрядных версиях, а длинный 32-разрядный (4 байта)

Полная ссылка на сигнатуры вызовов:

- Win32api32.txt для Visual Basic 5.0 (старые декларации API, последний раз рассмотренный в марте 2005 г., Microsoft)
- Win32API_PtrSafe с 64-разрядной поддержкой (Office 2010, Microsoft)

Examples

Объявление и использование API

Объявление процедуры DLL для работы с различными версиями VBA:

```
Option Explicit

#If Win64 Then

    Private Declare PtrSafe Sub xLib "Kernel32" Alias "Sleep" (ByVal dwMilliseconds As Long)

#ElseIf Win32 Then

    Private Declare Sub apiSleep Lib "Kernel32" Alias "Sleep" (ByVal dwMilliseconds As Long)

#End If
```

Вышеприведенное объявление указывает VBA, как вызвать функцию «Сон», определенную в файле Kernel32.dll

Win64 и Win32 - это предопределенные константы, используемые для условной компиляции

Предварительно определенные константы

Некоторые константы компиляции уже заранее определены. Какие из них будут зависеть от битности офисной версии, в которой вы используете VBA. Обратите внимание, что Vba7 был представлен вместе с Office 2010 для поддержки 64-разрядных версий Office.

постоянная	16 бит	32-битный	64-битный
VBA6	Ложь	Если Vba6	Ложь
Vba7	Ложь	Если Vba7	Правда
Win16	Правда	Ложь	Ложь
Win32	Ложь	Правда	Правда
Win64	Ложь	Ложь	Правда
макинтош	Ложь	Если Mac	Если Mac

Эти константы относятся к версии Office, а не к версии Windows. Например, Win32 = TRUE в 32-разрядном Office, даже если ОС - это 64-разрядная версия Windows.

Основное отличие при объявлении API-интерфейсов - от 32-битных до 64-битных версий

Office, которые ввели новые типы параметров (подробнее см. Раздел «Примечания»)

Заметки:

- Объявления размещаются в верхней части модуля и вне любых подписок или функций
- Процедуры, объявленные в стандартных модулях, общедоступны по умолчанию
- Чтобы объявить процедуру, закрытую для модуля, перед объявлением ключевым словом `Private`
- Процедуры DLL, объявленные в любом другом типе модуля, являются приватными для этого модуля

Простой пример для вызова API сна:

```
Public Sub TestPause()

    Dim start As Double

    start = Timer

    Sleep 9000      'Pause execution for 9 seconds

    Debug.Print "Paused for " & Format(Timer - start, "#,###.000") & " seconds"

    'Immediate window result: Paused for 9.000 seconds

End Sub
```

Рекомендуется создать выделенный модуль API для обеспечения легкого доступа к системным функциям из оберток VBA - обычных VBA Subs или функций, которые инкапсулируют детали, необходимые для фактического системного вызова, такие как параметры, используемые в библиотеках, и инициализацию этих параметров

Модуль может содержать все объявления и зависимости:

- Подписи методов и требуемые структуры данных
- Обертки, которые выполняют проверку ввода, и обеспечивают, чтобы все параметры передавались как ожидалось

Чтобы объявить процедуру DLL, добавьте оператор `Declare` в раздел Declarations окна кода.

Если процедура возвращает значение, объявите ее как **функцию** :

```
Declare Function publicname Lib "libname" [Alias "alias"] [([[ByVal] variable [As type]
```

```
[,[ByVal] variable [As type]]...])] As Type
```

Если процедура не возвращает значение, объявите его как **Sub** :

```
Declare Sub publicname Lib "libname" [Alias "alias"] [([[ByVal] variable [As type] [,[ByVal] variable [As type]]...])]
```

- !!! !! !!!!!!!!!!!!!!!!!!!!!!!!!

  Также следует отметить, что **большинство недействительных вызовов API приведет к сбою Excel** и, возможно, к поврежденным файлам данных

- !!! !! !!!!!!!!!!!!!!!!!!!!!!!!!

Office 2011 для Mac

```
Private Declare Function system Lib "libc.dylib" (ByVal command As String) As Long

Sub RunSafari()
    Dim result As Long
    result = system("open -a Safari --args http://www.google.com")
    Debug.Print Str(result)
End Sub
```

В приведенных ниже примерах (Windows API - выделенный модуль (1 и 2)) показан модуль API, который включает в себя общие объявления для Win64 и Win32

Windows API - выделенный модуль (1 из 2)

```
Option Explicit

#If Win64 Then  'Win64 = True, Win32 = False, Win16 = False
    Private Declare PtrSafe Sub apiCopyMemory Lib "Kernel32" Alias "RtlMoveMemory" (MyDest As Any, MySource As Any, ByVal MySize As Long)
    Private Declare PtrSafe Sub apiExitProcess Lib "Kernel32" Alias "ExitProcess" (ByVal uExitCode As Long)
    Private Declare PtrSafe Sub apiSetCursorPos Lib "User32" Alias "SetCursorPos" (ByVal X As Integer, ByVal Y As Integer)
    Private Declare PtrSafe Sub apiSleep Lib "Kernel32" Alias "Sleep" (ByVal dwMilliseconds As Long)
    Private Declare PtrSafe Function apiAttachThreadInput Lib "User32" Alias "AttachThreadInput" (ByVal idAttach As Long, ByVal idAttachTo As Long, ByVal fAttach As Long) As Long
    Private Declare PtrSafe Function apiBringWindowToTop Lib "User32" Alias "BringWindowToTop" (ByVal lngHWnd As Long) As Long
    Private Declare PtrSafe Function apiCloseWindow Lib "User32" Alias "CloseWindow" (ByVal hWnd As Long) As Long
    Private Declare PtrSafe Function apiDestroyWindow Lib "User32" Alias "DestroyWindow" (ByVal hWnd As Long) As Boolean
    Private Declare PtrSafe Function apiEndDialog Lib "User32" Alias "EndDialog" (ByVal hWnd As Long, ByVal result As Long) As Boolean
```

```
    Private Declare PtrSafe Function apiEnumChildWindows Lib "User32" Alias "EnumChildWindows"
(ByVal hWndParent As Long, ByVal pEnumProc As Long, ByVal lParam As Long) As Long
    Private Declare PtrSafe Function apiExitWindowsEx Lib "User32" Alias "ExitWindowsEx"
(ByVal uFlags As Long, ByVal dwReserved As Long) As Long
    Private Declare PtrSafe Function apiFindExecutable Lib "Shell32" Alias "FindExecutableA"
(ByVal lpFile As String, ByVallpDirectory As String, ByVal lpResult As String) As Long
    Private Declare PtrSafe Function apiFindWindow Lib "User32" Alias "FindWindowA" (ByVal
lpClassName As String, ByVal lpWindowName As String) As Long
    Private Declare PtrSafe Function apiFindWindowEx Lib "User32" Alias "FindWindowExA" (ByVal
hWnd1 As Long, ByVal hWnd2 As Long, ByVal lpsz1 As String, ByVal lpsz2 As String) As Long
    Private Declare PtrSafe Function apiGetActiveWindow Lib "User32" Alias "GetActiveWindow"
() As Long
    Private Declare PtrSafe Function apiGetClassNameA Lib "User32" Alias "GetClassNameA"
(ByVal hWnd As Long, ByVal szClassName As String, ByVal lLength As Long) As Long
    Private Declare PtrSafe Function apiGetCommandLine Lib "Kernel32" Alias "GetCommandLineW"
() As Long
    Private Declare PtrSafe Function apiGetCommandLineParams Lib "Kernel32" Alias
"GetCommandLineA" () As Long
    Private Declare PtrSafe Function apiGetDiskFreeSpaceEx Lib "Kernel32" Alias
"GetDiskFreeSpaceExA" (ByVal lpDirectoryName As String, lpFreeBytesAvailableToCaller As
Currency, lpTotalNumberOfBytes As Currency, lpTotalNumberOfFreeBytes As Currency) As Long
    Private Declare PtrSafe Function apiGetDriveType Lib "Kernel32" Alias "GetDriveTypeA"
(ByVal nDrive As String) As Long
    Private Declare PtrSafe Function apiGetExitCodeProcess Lib "Kernel32" Alias
"GetExitCodeProcess" (ByVal hProcess As Long, lpExitCode As Long) As Long
    Private Declare PtrSafe Function apiGetForegroundWindow Lib "User32" Alias
"GetForegroundWindow" () As Long
    Private Declare PtrSafe Function apiGetFrequency Lib "Kernel32" Alias
"QueryPerformanceFrequency" (cyFrequency As Currency) As Long
    Private Declare PtrSafe Function apiGetLastError Lib "Kernel32" Alias "GetLastError" () As
Integer
    Private Declare PtrSafe Function apiGetParent Lib "User32" Alias "GetParent" (ByVal hWnd
As Long) As Long
    Private Declare PtrSafe Function apiGetSystemMetrics Lib "User32" Alias "GetSystemMetrics"
(ByVal nIndex As Long) As Long
    Private Declare PtrSafe Function apiGetSystemMetrics32 Lib "User32" Alias
"GetSystemMetrics" (ByVal nIndex As Long) As Long
    Private Declare PtrSafe Function apiGetTickCount Lib "Kernel32" Alias
"QueryPerformanceCounter" (cyTickCount As Currency) As Long
    Private Declare PtrSafe Function apiGetTickCountMs Lib "Kernel32" Alias "GetTickCount" ()
As Long
    Private Declare PtrSafe Function apiGetUserName Lib "AdvApi32" Alias "GetUserNameA" (ByVal
lpBuffer As String, nSize As Long) As Long
    Private Declare PtrSafe Function apiGetWindow Lib "User32" Alias "GetWindow" (ByVal hWnd
As Long, ByVal wCmd As Long) As Long
    Private Declare PtrSafe Function apiGetWindowRect Lib "User32" Alias "GetWindowRect"
(ByVal hWnd As Long, lpRect As winRect) As Long
    Private Declare PtrSafe Function apiGetWindowText Lib "User32" Alias "GetWindowTextA"
(ByVal hWnd As Long, ByVal szWindowText As String, ByVal lLength As Long) As Long
    Private Declare PtrSafe Function apiGetWindowThreadProcessId Lib "User32" Alias
"GetWindowThreadProcessId" (ByVal hWnd As Long, lpdwProcessId As Long) As Long
    Private Declare PtrSafe Function apiIsCharAlphaNumericA Lib "User32" Alias
"IsCharAlphaNumericA" (ByVal byChar As Byte) As Long
    Private Declare PtrSafe Function apiIsIconic Lib "User32" Alias "IsIconic" (ByVal hWnd As
Long) As Long
    Private Declare PtrSafe Function apiIsWindowVisible Lib "User32" Alias "IsWindowVisible"
(ByVal hWnd As Long) As Long
    Private Declare PtrSafe Function apiIsZoomed Lib "User32" Alias "IsZoomed" (ByVal hWnd As
Long) As Long
    Private Declare PtrSafe Function apiLStrCpynA Lib "Kernel32" Alias "lstrcpynA" (ByVal
pDestination As String, ByVal pSource As Long, ByVal iMaxLength As Integer) As Long
```

```
    Private Declare PtrSafe Function apiMessageBox Lib "User32" Alias "MessageBoxA" (ByVal
hWnd As Long, ByVal lpText As String, ByVal lpCaption As String, ByVal wType As Long) As Long
    Private Declare PtrSafe Function apiOpenIcon Lib "User32" Alias "OpenIcon" (ByVal hWnd As
Long) As Long
    Private Declare PtrSafe Function apiOpenProcess Lib "Kernel32" Alias "OpenProcess" (ByVal
dwDesiredAccess As Long, ByVal bInheritHandle As Long, ByVal dwProcessId As Long) As Long
    Private Declare PtrSafe Function apiPathAddBackslashByPointer Lib "ShlwApi" Alias
"PathAddBackslashW" (ByVal lpszPath As Long) As Long
    Private Declare PtrSafe Function apiPathAddBackslashByString Lib "ShlwApi" Alias
"PathAddBackslashW" (ByVal lpszPath As String) As Long 'http://msdn.microsoft.com/en-
us/library/aa155716%28office.10%29.aspx
    Private Declare PtrSafe Function apiPostMessage Lib "User32" Alias "PostMessageA" (ByVal
hWnd As Long, ByVal wMsg As Long, ByVal wParam As Long, ByVal lParam As Long) As Long
    Private Declare PtrSafe Function apiRegQueryValue Lib "AdvApi32" Alias "RegQueryValue"
(ByVal hKey As Long, ByVal sValueName As String, ByVal dwReserved As Long, ByRef lValueType As
Long, ByVal sValue As String, ByRef lResultLen As Long) As Long
    Private Declare PtrSafe Function apiSendMessage Lib "User32" Alias "SendMessageA" (ByVal
hWnd As Long, ByVal wMsg As Long, ByVal wParam As Long, lParam As Any) As Long
    Private Declare PtrSafe Function apiSetActiveWindow Lib "User32" Alias "SetActiveWindow"
(ByVal hWnd As Long) As Long
    Private Declare PtrSafe Function apiSetCurrentDirectoryA Lib "Kernel32" Alias
"SetCurrentDirectoryA" (ByVal lpPathName As String) As Long
    Private Declare PtrSafe Function apiSetFocus Lib "User32" Alias "SetFocus" (ByVal hWnd As
Long) As Long
    Private Declare PtrSafe Function apiSetForegroundWindow Lib "User32" Alias
"SetForegroundWindow" (ByVal hWnd As Long) As Long
    Private Declare PtrSafe Function apiSetLocalTime Lib "Kernel32" Alias "SetLocalTime"
(lpSystem As SystemTime) As Long
    Private Declare PtrSafe Function apiSetWindowPlacement Lib "User32" Alias
"SetWindowPlacement" (ByVal hWnd As Long, ByRef lpwndpl As winPlacement) As Long
    Private Declare PtrSafe Function apiSetWindowPos Lib "User32" Alias "SetWindowPos" (ByVal
hWnd As Long, ByVal hWndInsertAfter As Long, ByVal X As Long, ByVal Y As Long, ByVal cx As
Long, ByVal cy As Long, ByVal wFlags As Long) As Long
    Private Declare PtrSafe Function apiSetWindowText Lib "User32" Alias "SetWindowTextA"
(ByVal hWnd As Long, ByVal lpString As String) As Long
    Private Declare PtrSafe Function apiShellExecute Lib "Shell32" 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
    Private Declare PtrSafe Function apiShowWindow Lib "User32" Alias "ShowWindow" (ByVal hWnd
As Long, ByVal nCmdShow As Long) As Long
    Private Declare PtrSafe Function apiShowWindowAsync Lib "User32" Alias "ShowWindowAsync"
(ByVal hWnd As Long, ByVal nCmdShow As Long) As Long
    Private Declare PtrSafe Function apiStrCpy Lib "Kernel32" Alias "lstrcpynA" (ByVal
pDestination As String, ByVal pSource As String, ByVal iMaxLength As Integer) As Long
    Private Declare PtrSafe Function apiStringLen Lib "Kernel32" Alias "lstrlenW" (ByVal
lpString As Long) As Long
    Private Declare PtrSafe Function apiStrTrimW Lib "ShlwApi" Alias "StrTrimW" () As Boolean
    Private Declare PtrSafe Function apiTerminateProcess Lib "Kernel32" Alias
"TerminateProcess" (ByVal hWnd As Long, ByVal uExitCode As Long) As Long
    Private Declare PtrSafe Function apiTimeGetTime Lib "Winmm" Alias "timeGetTime" () As Long
    Private Declare PtrSafe Function apiVarPtrArray Lib "MsVbVm50" Alias "VarPtr" (Var() As
Any) As Long
    Private Type browseInfo     'used by apiBrowseForFolder
        hOwner As Long
        pidlRoot As Long
        pszDisplayName As String
        lpszTitle As String
        ulFlags As Long
        lpfn As Long
        lParam As Long
        iImage As Long
```

```
    End Type
    Private Declare PtrSafe Function apiBrowseForFolder Lib "Shell32" Alias
"SHBrowseForFolderA" (lpBrowseInfo As browseInfo) As Long
    Private Type CHOOSECOLOR    'used by apiChooseColor;
http://support.microsoft.com/kb/153929 and http://www.cpearson.com/Excel/Colors.aspx
        lStructSize As Long
        hWndOwner As Long
        hInstance As Long
        rgbResult As Long
        lpCustColors As String
        flags As Long
        lCustData As Long
        lpfnHook As Long
        lpTemplateName As String
    End Type
    Private Declare PtrSafe Function apiChooseColor Lib "ComDlg32" Alias "ChooseColorA"
(pChoosecolor As CHOOSECOLOR) As Long
    Private Type FindWindowParameters   'Custom structure for passing in the parameters in/out
of the hook enumeration function; could use global variables instead, but this is nicer
        strTitle As String  'INPUT
        hWnd As Long        'OUTPUT
    End Type                            'Find a specific window with dynamic caption from a
list of all open windows: http://www.everythingaccess.com/tutorials.asp?ID=Bring-an-external-
application-window-to-the-foreground
    Private Declare PtrSafe Function apiEnumWindows Lib "User32" Alias "EnumWindows" (ByVal
lpEnumFunc As LongPtr, ByVal lParam As LongPtr) As Long
    Private Type lastInputInfo  'used by apiGetLastInputInfo, getLastInputTime
        cbSize As Long
        dwTime As Long
    End Type
    Private Declare PtrSafe Function apiGetLastInputInfo Lib "User32" Alias "GetLastInputInfo"
(ByRef plii As lastInputInfo) As Long
    'http://www.pgacon.com/visualbasic.htm#Take%20Advantage%20of%20Conditional%20Compilation
    'Logical and Bitwise Operators in Visual Basic: http://msdn.microsoft.com/en-
us/library/wz3k228a(v=vs.80).aspx and http://stackoverflow.com/questions/1070863/hidden-
features-of-vba
    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 Declare PtrSafe Sub apiGetLocalTime Lib "Kernel32" Alias "GetLocalTime" (lpSystem
As SystemTime)
    Private Type pointAPI       'used by apiSetWindowPlacement
         X As Long
         Y As Long
    End Type
    Private Type rectAPI       'used by apiSetWindowPlacement
        Left_Renamed As Long
        Top_Renamed As Long
        Right_Renamed As Long
        Bottom_Renamed As Long
    End Type
    Private Type winPlacement   'used by apiSetWindowPlacement
        length As Long
        flags As Long
```

```
        showCmd As Long
        ptMinPosition As pointAPI
        ptMaxPosition As pointAPI
        rcNormalPosition As rectAPI
    End Type
    Private Declare PtrSafe Function apiGetWindowPlacement Lib "User32" Alias
"GetWindowPlacement" (ByVal hWnd As Long, ByRef lpwndpl As winPlacement) As Long
    Private Type winRect     'used by apiMoveWindow
        Left As Long
        Top As Long
        Right As Long
        Bottom As Long
    End Type
    Private Declare PtrSafe Function apiMoveWindow Lib "User32" Alias "MoveWindow" (ByVal hWnd
As Long, xLeft As Long, ByVal yTop As Long, wWidth As Long, ByVal hHeight As Long, ByVal
repaint As Long) As Long

    Private Declare PtrSafe Function apiInternetOpen Lib "WiniNet" Alias "InternetOpenA"
(ByVal sAgent As String, ByVal lAccessType As Long, ByVal sProxyName As String, ByVal
sProxyBypass As String, ByVal lFlags As Long) As Long    'Open the Internet object    'ex:
lngINet = InternetOpen("MyFTP Control", 1, vbNullString, vbNullString, 0)
    Private Declare PtrSafe Function apiInternetConnect Lib "WiniNet" Alias "InternetConnectA"
(ByVal hInternetSession As Long, ByVal sServerName As String, ByVal nServerPort As Integer,
ByVal sUsername As String, ByVal sPassword As String, ByVal lService As Long, ByVal lFlags As
Long, ByVal lContext As Long) As Long  'Connect to the network  'ex: lngINetConn =
InternetConnect(lngINet, "ftp.microsoft.com", 0, "anonymous", "wally@wallyworld.com", 1, 0, 0)
    Private Declare PtrSafe Function apiFtpGetFile Lib "WiniNet" Alias "FtpGetFileA" (ByVal
hFtpSession As Long, ByVal lpszRemoteFile As String, ByVal lpszNewFile As String, ByVal
fFailIfExists As Boolean, ByVal dwFlagsAndAttributes As Long, ByVal dwFlags As Long, ByVal
dwContext As Long) As Boolean    'Get a file 'ex: blnRC = FtpGetFile(lngINetConn,
"dirmap.txt", "c:\dirmap.txt", 0, 0, 1, 0)
    Private Declare PtrSafe Function apiFtpPutFile Lib "WiniNet" Alias "FtpPutFileA" (ByVal
hFtpSession As Long, ByVal lpszLocalFile As String, ByVal lpszRemoteFile As String, ByVal
dwFlags As Long, ByVal dwContext As Long) As Boolean  'Send a file  'ex: blnRC =
FtpPutFile(lngINetConn, "c:\dirmap.txt", "dirmap.txt", 1, 0)
    Private Declare PtrSafe Function apiFtpDeleteFile Lib "WiniNet" Alias "FtpDeleteFileA"
(ByVal hFtpSession As Long, ByVal lpszFileName As String) As Boolean 'Delete a file 'ex: blnRC
= FtpDeleteFile(lngINetConn, "test.txt")
    Private Declare PtrSafe Function apiInternetCloseHandle Lib "WiniNet" (ByVal hInet As
Long) As Integer  'Close the Internet object  'ex: InternetCloseHandle lngINetConn    'ex:
InternetCloseHandle lngINet
    Private Declare PtrSafe Function apiFtpFindFirstFile Lib "WiniNet" Alias
"FtpFindFirstFileA" (ByVal hFtpSession As Long, ByVal lpszSearchFile As String, lpFindFileData
As WIN32_FIND_DATA, ByVal dwFlags As Long, ByVal dwContent As Long) As Long
    Private Type FILETIME
        dwLowDateTime As Long
        dwHighDateTime As Long
    End Type
    Private Type WIN32_FIND_DATA
        dwFileAttributes As Long
        ftCreationTime As FILETIME
        ftLastAccessTime As FILETIME
        ftLastWriteTime As FILETIME
        nFileSizeHigh As Long
        nFileSizeLow As Long
        dwReserved0 As Long
        dwReserved1 As Long
        cFileName As String * 1 'MAX_FTP_PATH
        cAlternate As String * 14
    End Type    'ex: lngHINet = FtpFindFirstFile(lngINetConn, "*.*", pData, 0, 0)
    Private Declare PtrSafe Function apiInternetFindNextFile Lib "WiniNet" Alias
```

```
"InternetFindNextFileA" (ByVal hFind As Long, lpvFindData As WIN32_FIND_DATA) As Long  'ex:
blnRC = InternetFindNextFile(lngHINet, pData)
#ElseIf Win32 Then  'Win32 = True, Win16 = False
```

(продолжение во втором примере)

Windows API - выделенный модуль (2 из 2)

```
#ElseIf Win32 Then  'Win32 = True, Win16 = False
    Private Declare Sub apiCopyMemory Lib "Kernel32" Alias "RtlMoveMemory" (MyDest As Any,
MySource As Any, ByVal MySize As Long)
    Private Declare Sub apiExitProcess Lib "Kernel32" Alias "ExitProcess" (ByVal uExitCode As
Long)
    'Private Declare Sub apiGetStartupInfo Lib "Kernel32" Alias "GetStartupInfoA"
(lpStartupInfo As STARTUPINFO)
    Private Declare Sub apiSetCursorPos Lib "User32" Alias "SetCursorPos" (ByVal X As Integer,
ByVal Y As Integer)  'Logical and Bitwise Operators in Visual Basic:
http://msdn.microsoft.com/en-us/library/wz3k228a(v=vs.80).aspx and
http://stackoverflow.com/questions/1070863/hidden-features-of-vba
'http://www.pgacon.com/visualbasic.htm#Take%20Advantage%20of%20Conditional%20Compilation
    Private Declare Sub apiSleep Lib "Kernel32" Alias "Sleep" (ByVal dwMilliseconds As Long)
    Private Declare Function apiAttachThreadInput Lib "User32" Alias "AttachThreadInput"
(ByVal idAttach As Long, ByVal idAttachTo As Long, ByVal fAttach As Long) As Long
    Private Declare Function apiBringWindowToTop Lib "User32" Alias "BringWindowToTop" (ByVal
lngHWnd As Long) As Long
    Private Declare Function apiCloseHandle Lib "Kernel32" (ByVal hObject As Long) As Long
    Private Declare Function apiCloseWindow Lib "User32" Alias "CloseWindow" (ByVal hWnd As
Long) As Long
    'Private Declare Function apiCreatePipe Lib "Kernel32" (phReadPipe As Long, phWritePipe As
Long, lpPipeAttributes As SECURITY_ATTRIBUTES, ByVal nSize As Long) As Long
    'Private Declare Function apiCreateProcess Lib "Kernel32" Alias "CreateProcessA" (ByVal
lpApplicationName As Long, ByVal lpCommandLine As String, lpProcessAttributes As Any,
lpThreadAttributes As Any, ByVal bInheritHandles As Long, ByVal dwCreationFlags As Long,
lpEnvironment As Any, ByVal lpCurrentDriectory As String, lpStartupInfo As STARTUPINFO,
lpProcessInformation As PROCESS_INFORMATION) As Long
    Private Declare Function apiDestroyWindow Lib "User32" Alias "DestroyWindow" (ByVal hWnd
As Long) As Boolean
    Private Declare Function apiEndDialog Lib "User32" Alias "EndDialog" (ByVal hWnd As Long,
ByVal result As Long) As Boolean
    Private Declare Function apiEnumChildWindows Lib "User32" Alias "EnumChildWindows" (ByVal
hWndParent As Long, ByVal pEnumProc As Long, ByVal lParam As Long) As Long
    Private Declare Function apiExitWindowsEx Lib "User32" Alias "ExitWindowsEx" (ByVal uFlags
As Long, ByVal dwReserved As Long) As Long
    Private Declare Function apiFindExecutable Lib "Shell32" Alias "FindExecutableA" (ByVal
lpFile As String, ByVallpDirectory As String, ByVal lpResult As String) As Long
    Private Declare Function apiFindWindow Lib "User32" Alias "FindWindowA" (ByVal lpClassName
As String, ByVal lpWindowName As String) As Long
    Private Declare Function apiFindWindowEx Lib "User32" Alias "FindWindowExA" (ByVal hWnd1
As Long, ByVal hWnd2 As Long, ByVal lpsz1 As String, ByVal lpsz2 As String) As Long
    Private Declare Function apiGetActiveWindow Lib "User32" Alias "GetActiveWindow" () As
Long
    Private Declare Function apiGetClassNameA Lib "User32" Alias "GetClassNameA" (ByVal hWnd
As Long, ByVal szClassName As String, ByVal lLength As Long) As Long
    Private Declare Function apiGetCommandLine Lib "Kernel32" Alias "GetCommandLineW" () As
Long
    Private Declare Function apiGetCommandLineParams Lib "Kernel32" Alias "GetCommandLineA" ()
As Long
    Private Declare Function apiGetDiskFreeSpaceEx Lib "Kernel32" Alias "GetDiskFreeSpaceExA"
(ByVal lpDirectoryName As String, lpFreeBytesAvailableToCaller As Currency,
```

```
lpTotalNumberOfBytes As Currency, lpTotalNumberOfFreeBytes As Currency) As Long
    Private Declare Function apiGetDriveType Lib "Kernel32" Alias "GetDriveTypeA" (ByVal
nDrive As String) As Long
    Private Declare Function apiGetExitCodeProcess Lib "Kernel32" (ByVal hProcess As Long,
lpExitCode As Long) As Long
    Private Declare Function apiGetFileSize Lib "Kernel32" (ByVal hFile As Long,
lpFileSizeHigh As Long) As Long
    Private Declare Function apiGetForegroundWindow Lib "User32" Alias "GetForegroundWindow"
() As Long
    Private Declare Function apiGetFrequency Lib "Kernel32" Alias "QueryPerformanceFrequency"
(cyFrequency As Currency) As Long
    Private Declare Function apiGetLastError Lib "Kernel32" Alias "GetLastError" () As Integer
    Private Declare Function apiGetParent Lib "User32" Alias "GetParent" (ByVal hWnd As Long)
As Long
    Private Declare Function apiGetSystemMetrics Lib "User32" Alias "GetSystemMetrics" (ByVal
nIndex As Long) As Long
    Private Declare Function apiGetTickCount Lib "Kernel32" Alias "QueryPerformanceCounter"
(cyTickCount As Currency) As Long
    Private Declare Function apiGetTickCountMs Lib "Kernel32" Alias "GetTickCount" () As Long
    Private Declare Function apiGetUserName Lib "AdvApi32" Alias "GetUserNameA" (ByVal
lpBuffer As String, nSize As Long) As Long
    Private Declare Function apiGetWindow Lib "User32" Alias "GetWindow" (ByVal hWnd As Long,
ByVal wCmd As Long) As Long
    Private Declare Function apiGetWindowRect Lib "User32" Alias "GetWindowRect" (ByVal hWnd
As Long, lpRect As winRect) As Long
    Private Declare Function apiGetWindowText Lib "User32" Alias "GetWindowTextA" (ByVal hWnd
As Long, ByVal szWindowText As String, ByVal lLength As Long) As Long
    Private Declare Function apiGetWindowThreadProcessId Lib "User32" Alias
"GetWindowThreadProcessId" (ByVal hWnd As Long, lpdwProcessId As Long) As Long
    Private Declare Function apiIsCharAlphaNumericA Lib "User32" Alias "IsCharAlphaNumericA"
(ByVal byChar As Byte) As Long
    Private Declare Function apiIsIconic Lib "User32" Alias "IsIconic" (ByVal hWnd As Long) As
Long
    Private Declare Function apiIsWindowVisible Lib "User32" Alias "IsWindowVisible" (ByVal
hWnd As Long) As Long
    Private Declare Function apiIsZoomed Lib "User32" Alias "IsZoomed" (ByVal hWnd As Long) As
Long
    Private Declare Function apiLStrCpynA Lib "Kernel32" Alias "lstrcpynA" (ByVal pDestination
As String, ByVal pSource As Long, ByVal iMaxLength As Integer) As Long
    Private Declare Function apiMessageBox Lib "User32" Alias "MessageBoxA" (ByVal hWnd As
Long, ByVal lpText As String, ByVal lpCaption As String, ByVal wType As Long) As Long
    Private Declare Function apiOpenIcon Lib "User32" Alias "OpenIcon" (ByVal hWnd As Long) As
Long
    Private Declare Function apiOpenProcess Lib "Kernel32" Alias "OpenProcess" (ByVal
dwDesiredAccess As Long, ByVal bInheritHandle As Long, ByVal dwProcessId As Long) As Long
    Private Declare Function apiPathAddBackslashByPointer Lib "ShlwApi" Alias
"PathAddBackslashW" (ByVal lpszPath As Long) As Long
    Private Declare Function apiPathAddBackslashByString Lib "ShlwApi" Alias
"PathAddBackslashW" (ByVal lpszPath As String) As Long 'http://msdn.microsoft.com/en-
us/library/aa155716%28office.10%29.aspx
    Private Declare Function apiPostMessage Lib "User32" Alias "PostMessageA" (ByVal hWnd As
Long, ByVal wMsg As Long, ByVal wParam As Long, ByVal lParam As Long) As Long
    Private Declare Function apiReadFile Lib "Kernel32" (ByVal hFile As Long, lpBuffer As Any,
ByVal nNumberOfBytesToRead As Long, lpNumberOfBytesRead As Long, lpOverlapped As Any) As Long
    Private Declare Function apiRegQueryValue Lib "AdvApi32" Alias "RegQueryValue" (ByVal hKey
As Long, ByVal sValueName As String, ByVal dwReserved As Long, ByRef lValueType As Long, ByVal
sValue As String, ByRef lResultLen As Long) As Long
    Private Declare Function apiSendMessage Lib "User32" Alias "SendMessageA" (ByVal hWnd As
Long, ByVal wMsg As Long, ByVal wParam As Long, lParam As Any) As Long
    Private Declare Function apiSetActiveWindow Lib "User32" Alias "SetActiveWindow" (ByVal
hWnd As Long) As Long
```

```
    Private Declare Function apiSetCurrentDirectoryA Lib "Kernel32" Alias
"SetCurrentDirectoryA" (ByVal lpPathName As String) As Long
    Private Declare Function apiSetFocus Lib "User32" Alias "SetFocus" (ByVal hWnd As Long) As
Long
    Private Declare Function apiSetForegroundWindow Lib "User32" Alias "SetForegroundWindow"
(ByVal hWnd As Long) As Long
    Private Declare Function apiSetLocalTime Lib "Kernel32" Alias "SetLocalTime" (lpSystem As
SystemTime) As Long
    Private Declare Function apiSetWindowPlacement Lib "User32" Alias "SetWindowPlacement"
(ByVal hWnd As Long, ByRef lpwndpl As winPlacement) As Long
    Private Declare Function apiSetWindowPos Lib "User32" Alias "SetWindowPos" (ByVal hWnd As
Long, ByVal hWndInsertAfter As Long, ByVal X As Long, ByVal Y As Long, ByVal cx As Long, ByVal
cy As Long, ByVal wFlags As Long) As Long
    Private Declare Function apiSetWindowText Lib "User32" Alias "SetWindowTextA" (ByVal hWnd
As Long, ByVal lpString As String) As Long
    Private Declare Function apiShellExecute Lib "Shell32" 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
    Private Declare Function apiShowWindow Lib "User32" Alias "ShowWindow" (ByVal hWnd As
Long, ByVal nCmdShow As Long) As Long
    Private Declare Function apiShowWindowAsync Lib "User32" Alias "ShowWindowAsync" (ByVal
hWnd As Long, ByVal nCmdShow As Long) As Long
    Private Declare Function apiStrCpy Lib "Kernel32" Alias "lstrcpynA" (ByVal pDestination As
String, ByVal pSource As String, ByVal iMaxLength As Integer) As Long
    Private Declare Function apiStringLen Lib "Kernel32" Alias "lstrlenW" (ByVal lpString As
Long) As Long
    Private Declare Function apiStrTrimW Lib "ShlwApi" Alias "StrTrimW" () As Boolean
    Private Declare Function apiTerminateProcess Lib "Kernel32" Alias "TerminateProcess"
(ByVal hWnd As Long, ByVal uExitCode As Long) As Long
    Private Declare Function apiTimeGetTime Lib "Winmm" Alias "timeGetTime" () As Long
    Private Declare Function apiVarPtrArray Lib "MsVbVm50" Alias "VarPtr" (Var() As Any) As
Long
    Private Declare Function apiWaitForSingleObject Lib "Kernel32" (ByVal hHandle As Long,
ByVal dwMilliseconds As Long) As Long
    Private Type browseInfo     'used by apiBrowseForFolder
        hOwner As Long
        pidlRoot As Long
        pszDisplayName As String
        lpszTitle As String
        ulFlags As Long
        lpfn As Long
        lParam As Long
        iImage As Long
    End Type
    Private Declare Function apiBrowseForFolder Lib "Shell32" Alias "SHBrowseForFolderA"
(lpBrowseInfo As browseInfo) As Long
    Private Type CHOOSECOLOR    'used by apiChooseColor;
http://support.microsoft.com/kb/153929 and http://www.cpearson.com/Excel/Colors.aspx
        lStructSize As Long
        hWndOwner As Long
        hInstance As Long
        rgbResult As Long
        lpCustColors As String
        flags As Long
        lCustData As Long
        lpfnHook As Long
        lpTemplateName As String
    End Type
    Private Declare Function apiChooseColor Lib "ComDlg32" Alias "ChooseColorA" (pChoosecolor
As CHOOSECOLOR) As Long
    Private Type FindWindowParameters   'Custom structure for passing in the parameters in/out
```

```
of the hook enumeration function; could use global variables instead, but this is nicer
        strTitle As String  'INPUT
        hWnd As Long        'OUTPUT
    End Type                            'Find a specific window with dynamic caption from a
list of all open windows: http://www.everythingaccess.com/tutorials.asp?ID=Bring-an-external-
application-window-to-the-foreground
    Private Declare Function apiEnumWindows Lib "User32" Alias "EnumWindows" (ByVal lpEnumFunc
As Long, ByVal lParam As Long) As Long
    Private Type lastInputInfo  'used by apiGetLastInputInfo, getLastInputTime
        cbSize As Long
        dwTime As Long
    End Type
    Private Declare Function apiGetLastInputInfo Lib "User32" Alias "GetLastInputInfo" (ByRef
plii As lastInputInfo) As Long
    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 Declare Sub apiGetLocalTime Lib "Kernel32" Alias "GetLocalTime" (lpSystem As
SystemTime)
    Private Type pointAPI
        X As Long
        Y As Long
    End Type
    Private Type rectAPI
        Left_Renamed As Long
        Top_Renamed As Long
        Right_Renamed As Long
        Bottom_Renamed As Long
    End Type
    Private Type winPlacement
        length As Long
        flags As Long
        showCmd As Long
        ptMinPosition As pointAPI
        ptMaxPosition As pointAPI
        rcNormalPosition As rectAPI
    End Type
    Private Declare Function apiGetWindowPlacement Lib "User32" Alias "GetWindowPlacement"
(ByVal hWnd As Long, ByRef lpwndpl As winPlacement) As Long
    Private Type winRect
        Left As Long
        Top As Long
        Right As Long
        Bottom As Long
    End Type
    Private Declare Function apiMoveWindow Lib "User32" Alias "MoveWindow" (ByVal hWnd As
Long, xLeft As Long, ByVal yTop As Long, wWidth As Long, ByVal hHeight As Long, ByVal repaint
As Long) As Long
#Else   ' Win16 = True
#End If
```

Mac API

Microsoft официально не поддерживает API, но с некоторыми исследованиями больше объявлений можно найти в Интернете

Office 2016 для Mac изолирован песочницей

В отличие от других версий приложений Office, поддерживающих VBA, приложения Office 2016 для Mac изолированы.

Песочница ограничивает доступ приложений к ресурсам за пределами контейнера приложения. Это влияет на любые надстройки или макросы, которые связаны с доступом к файлам или связью между процессами. Вы можете минимизировать эффекты песочницы, используя новые команды, описанные в следующем разделе. Новые команды VBA для Office 2016 для Mac

Следующие команды VBA являются новыми и уникальными для Office 2016 для Mac.

команда	Используйте для
GrantAccessToMultipleFiles	Запросить разрешение пользователя на доступ к нескольким файлам одновременно
AppleScriptTask	Вызовите внешние скрипты AppleScript от VB
MAC_OFFICE_VERSION	IFDEF между различными версиями Mac Office во время компиляции

Office 2011 для Mac

```
Private Declare Function system Lib "libc.dylib" (ByVal command As String) As Long
Private Declare Function popen Lib "libc.dylib" (ByVal command As String, ByVal mode As
String) As Long
Private Declare Function pclose Lib "libc.dylib" (ByVal file As Long) As Long
Private Declare Function fread Lib "libc.dylib" (ByVal outStr As String, ByVal size As Long,
ByVal items As Long, ByVal stream As Long) As Long
Private Declare Function feof Lib "libc.dylib" (ByVal file As Long) As Long
```

-

Office 2016 для Mac

```
Private Declare PtrSafe Function popen Lib "libc.dylib" (ByVal command As String, ByVal mode
As String) As LongPtr
Private Declare PtrSafe Function pclose Lib "libc.dylib" (ByVal file As LongPtr) As Long
Private Declare PtrSafe Function fread Lib "libc.dylib" (ByVal outStr As String, ByVal size As
LongPtr, ByVal items As LongPtr, ByVal stream As LongPtr) As Long
Private Declare PtrSafe Function feof Lib "libc.dylib" (ByVal file As LongPtr) As LongPtr
```

Получить общие мониторы и разрешение экрана

```
Option Explicit

'GetSystemMetrics32 info: http://msdn.microsoft.com/en-us/library/ms724385(VS.85).aspx
#If Win64 Then
    Private Declare PtrSafe Function GetSystemMetrics32 Lib "User32" Alias "GetSystemMetrics"
(ByVal nIndex As Long) As Long
#ElseIf Win32 Then
    Private Declare Function GetSystemMetrics32 Lib "User32" Alias "GetSystemMetrics" (ByVal
nIndex As Long) As Long
#End If

'VBA Wrappers:
Public Function dllGetMonitors() As Long
    Const SM_CMONITORS = 80
    dllGetMonitors = GetSystemMetrics32(SM_CMONITORS)
End Function

Public Function dllGetHorizontalResolution() As Long
    Const SM_CXVIRTUALSCREEN = 78
    dllGetHorizontalResolution = GetSystemMetrics32(SM_CXVIRTUALSCREEN)
End Function

Public Function dllGetVerticalResolution() As Long
    Const SM_CYVIRTUALSCREEN = 79
    dllGetVerticalResolution = GetSystemMetrics32(SM_CYVIRTUALSCREEN)
End Function

Public Sub ShowDisplayInfo()
    Debug.Print "Total monitors: " & vbTab & vbTab & dllGetMonitors
    Debug.Print "Horizontal Resolution: " & vbTab & dllGetHorizontalResolution
    Debug.Print "Vertical Resolution: " & vbTab & dllGetVerticalResolution

    'Total monitors:         1
    'Horizontal Resolution:  1920
    'Vertical Resolution:    1080
End Sub
```

FTP и региональные API

modFTP

```
Option Explicit
Option Compare Text
Option Private Module

'http://msdn.microsoft.com/en-us/library/aa384180(v=VS.85).aspx
'http://www.dailydoseofexcel.com/archives/2006/01/29/ftp-via-vba/
'http://www.15seconds.com/issue/981203.htm

'Open the Internet object
Private Declare Function InternetOpen Lib "wininet.dll" Alias "InternetOpenA" ( _
    ByVal sAgent As String, _
    ByVal lAccessType As Long, _
    ByVal sProxyName As String, _
    ByVal sProxyBypass As String, _
```

```
    ByVal lFlags As Long _
) As Long
'ex: lngINet = InternetOpen("MyFTP Control", 1, vbNullString, vbNullString, 0)

'Connect to the network
Private Declare Function InternetConnect Lib "wininet.dll" Alias "InternetConnectA" ( _
    ByVal hInternetSession As Long, _
    ByVal sServerName As String, _
    ByVal nServerPort As Integer, _
    ByVal sUsername As String, _
    ByVal sPassword As String, _
    ByVal lService As Long, _
    ByVal lFlags As Long, _
    ByVal lContext As Long _
) As Long
'ex: lngINetConn = InternetConnect(lngINet, "ftp.microsoft.com", 0, "anonymous",
"wally@wallyworld.com", 1, 0, 0)

'Get a file
Private Declare Function FtpGetFile Lib "wininet.dll" Alias "FtpGetFileA" ( _
    ByVal hFtpSession As Long, _
    ByVal lpszRemoteFile As String, _
    ByVal lpszNewFile As String, _
    ByVal fFailIfExists As Boolean, _
    ByVal dwFlagsAndAttributes As Long, _
    ByVal dwFlags As Long, _
    ByVal dwContext As Long _
) As Boolean
'ex: blnRC = FtpGetFile(lngINetConn, "dirmap.txt", "c:\dirmap.txt", 0, 0, 1, 0)

'Send a file
Private Declare Function FtpPutFile Lib "wininet.dll" Alias "FtpPutFileA" _
( _
    ByVal hFtpSession As Long, _
    ByVal lpszLocalFile As String, _
    ByVal lpszRemoteFile As String, _
    ByVal dwFlags As Long, ByVal dwContext As Long _
) As Boolean
'ex: blnRC = FtpPutFile(lngINetConn, "c:\dirmap.txt", "dirmap.txt", 1, 0)

'Delete a file
Private Declare Function FtpDeleteFile Lib "wininet.dll" Alias "FtpDeleteFileA" _
( _
    ByVal hFtpSession As Long, _
    ByVal lpszFileName As String _
) As Boolean
'ex: blnRC = FtpDeleteFile(lngINetConn, "test.txt")

'Close the Internet object
Private Declare Function InternetCloseHandle Lib "wininet.dll" (ByVal hInet As Long) As
Integer
'ex: InternetCloseHandle lngINetConn
'ex: InternetCloseHandle lngINet



Private Declare Function FtpFindFirstFile Lib "wininet.dll" Alias "FtpFindFirstFileA" _
( _
    ByVal hFtpSession As Long, _
    ByVal lpszSearchFile As String, _
    lpFindFileData As WIN32_FIND_DATA, _
```

```
    ByVal dwFlags As Long, _
    ByVal dwContent As Long _
) As Long
Private Type FILETIME
        dwLowDateTime As Long
        dwHighDateTime As Long
End Type
Private Type WIN32_FIND_DATA
        dwFileAttributes As Long
        ftCreationTime As FILETIME
        ftLastAccessTime As FILETIME
        ftLastWriteTime As FILETIME
        nFileSizeHigh As Long
        nFileSizeLow As Long
        dwReserved0 As Long
        dwReserved1 As Long
        cFileName As String * MAX_FTP_PATH
        cAlternate As String * 14
End Type
'ex: lngHINet = FtpFindFirstFile(lngINetConn, "*.*", pData, 0, 0)

Private Declare Function InternetFindNextFile Lib "wininet.dll" Alias "InternetFindNextFileA"
_
( _
    ByVal hFind As Long, _
    lpvFindData As WIN32_FIND_DATA _
) As Long
'ex: blnRC = InternetFindNextFile(lngHINet, pData)


Public Sub showLatestFTPVersion()
    Dim ftpSuccess As Boolean, msg As String, lngFindFirst As Long
    Dim lngINet As Long, lngINetConn As Long
    Dim pData As WIN32_FIND_DATA
    'init the filename buffer
    pData.cFileName = String(260, 0)

    msg = "FTP Error"
    lngINet = InternetOpen("MyFTP Control", 1, vbNullString, vbNullString, 0)
    If lngINet > 0 Then
        lngINetConn = InternetConnect(lngINet, FTP_SERVER_NAME, FTP_SERVER_PORT,
FTP_USER_NAME, FTP_PASSWORD, 1, 0, 0)
        If lngINetConn > 0 Then
                FtpPutFile lngINetConn, "C:\Tmp\ftp.cls", "ftp.cls", FTP_TRANSFER_BINARY, 0
                'lngFindFirst = FtpFindFirstFile(lngINetConn, "ExcelDiff.xlsm", pData, 0, 0)
            If lngINet = 0 Then
                msg = "DLL error: " & Err.LastDllError & ", Error Number: " & Err.Number & ",
Error Desc: " & Err.Description
            Else
                msg = left(pData.cFileName, InStr(1, pData.cFileName, String(1, 0),
vbBinaryCompare) - 1)
            End If
            InternetCloseHandle lngINetConn
        End If
    InternetCloseHandle lngINet
    End If
        MsgBox msg
End Sub
```

---

modRegional:

```
Option Explicit

Private Const LOCALE_SDECIMAL = &HE
Private Const LOCALE_SLIST = &HC

Private Declare Function GetLocaleInfo Lib "Kernel32" Alias "GetLocaleInfoA" (ByVal Locale As
Long, ByVal LCType As Long, ByVal lpLCData As String, ByVal cchData As Long) As Long
Private Declare Function SetLocaleInfo Lib "Kernel32" Alias "SetLocaleInfoA" (ByVal Locale As
Long, ByVal LCType As Long, ByVal lpLCData As String) As Boolean
Private Declare Function GetUserDefaultLCID% Lib "Kernel32" ()

Public Function getTimeSeparator() As String
    getTimeSeparator = Application.International(xlTimeSeparator)
End Function
Public Function getDateSeparator() As String
    getDateSeparator = Application.International(xlDateSeparator)
End Function
Public Function getListSeparator() As String
    Dim ListSeparator As String, iRetVal1 As Long, iRetVal2 As Long, lpLCDataVar As String,
Position As Integer, Locale As Long
    Locale = GetUserDefaultLCID()
    iRetVal1 = GetLocaleInfo(Locale, LOCALE_SLIST, lpLCDataVar, 0)
    ListSeparator = String$(iRetVal1, 0)
    iRetVal2 = GetLocaleInfo(Locale, LOCALE_SLIST, ListSeparator, iRetVal1)
    Position = InStr(ListSeparator, Chr$(0))
    If Position > 0 Then ListSeparator = Left$(ListSeparator, Position - 1) Else ListSeparator
= vbNullString
    getListSeparator = ListSeparator
End Function

Private Sub ChangeSettingExample()  'change the setting of the character displayed as the
decimal separator.
    Call SetLocalSetting(LOCALE_SDECIMAL, ",")  'to change to ","
    Stop                                        'check your control panel to verify or use the
GetLocaleInfo API function
    Call SetLocalSetting(LOCALE_SDECIMAL, ".")  'to back change to "."
End Sub

Private Function SetLocalSetting(LC_CONST As Long, Setting As String) As Boolean
    Call SetLocaleInfo(GetUserDefaultLCID(), LC_CONST, Setting)
End Function
```

Прочитайте API-запросы онлайн: https://riptutorial.com/ru/vba/topic/10569/api-запросы

глава 3: CreateObject vs. GetObject

замечания

В своем простейшем случае `CreateObject` создает экземпляр объекта, тогда как `GetObject` получает существующий экземпляр объекта. Определение того, может ли объект быть создан или получен, будет зависеть от его свойства Instancing . Некоторые объекты SingleUse (например, WMI) и не могут быть созданы, если они уже существуют. Другие объекты (например, Excel) - это MultiUse и позволяют запускать сразу несколько экземпляров. Если экземпляр объекта еще не существует и вы `GetObject` , вы получите следующее сообщение `Run-time error '429': ActiveX component can't create object` .

GetObject требует наличия хотя бы одного из этих двух необязательных параметров:

1. *Pathname* - Variant (String): полный путь, включая имя файла, файла, содержащего объект. Этот параметр является необязательным, но *Class* требуется, если *Pathname* опущено.
2. *Class* - Variant (String): Строка, представляющая формальное определение (Application и ObjectType) объекта. *Класс* требуется, если *Pathname* опущен.

CreateObject имеет один обязательный параметр и один необязательный параметр:

1. *Class* - Variant (String): Строка, представляющая формальное определение (Application и ObjectType) объекта. *Класс* является обязательным параметром.
2. *Servername* - Variant (String): имя удаленного компьютера, на котором будет создан объект. Если этот параметр опущен, объект будет создан на локальном компьютере.

Класс всегда состоит из двух частей в виде `Application.ObjectType` :

1. *Приложение* - имя приложения, частью которого является объект. |
2. *Тип объекта* - тип создаваемого объекта. |

Некоторые примеры классов:

1. Word.Application
2. Лист Excel
3. Scripting.FileSystemObject

Examples

Демонстрация GetObject и CreateObject

Функция MSDN-GetObject

Возвращает ссылку на объект, предоставленный компонентом ActiveX.

Используйте функцию GetObject, когда есть текущий экземпляр объекта, или если вы хотите создать объект с уже загруженным файлом. Если текущий экземпляр отсутствует, и вы не хотите, чтобы объект запускался с загруженным файлом, используйте функцию CreateObject.

```
Sub CreateVSGet()
    Dim ThisXLApp As Excel.Application 'An example of early binding
    Dim AnotherXLApp As Object 'An example of late binding
    Dim ThisNewWB As Workbook
    Dim AnotherNewWB As Workbook
    Dim wb As Workbook

    'Get this instance of Excel
    Set ThisXLApp = GetObject(ThisWorkbook.Name).Application
    'Create another instance of Excel
    Set AnotherXLApp = CreateObject("Excel.Application")
    'Make the 2nd instance visible
    AnotherXLApp.Visible = True
    'Add a workbook to the 2nd instance
    Set AnotherNewWB = AnotherXLApp.Workbooks.Add
    'Add a sheet to the 2nd instance
    AnotherNewWB.Sheets.Add

    'You should now have 2 instances of Excel open
    'The 1st instance has 1 workbook: Book1
    'The 2nd instance has 1 workbook: Book2

    'Lets add another workbook to our 1st instance
    Set ThisNewWB = ThisXLApp.Workbooks.Add
    'Now loop through the workbooks and show their names
    For Each wb In ThisXLApp.Workbooks
        Debug.Print wb.Name
    Next
    'Now the 1st instance has 2 workbooks: Book1 and Book3
    'If you close the first instance of Excel,
    'Book1 and Book3 will close, but book2 will still be open

End Sub
```

Прочитайте CreateObject vs. GetObject онлайн:
https://riptutorial.com/ru/vba/topic/7729/createobject-vs--getobject

глава 4: Scripting.FileSystemObject

Examples

Создание файла FileSystemObject

```
Const ForReading = 1
Const ForWriting = 2
Const ForAppending = 8

Sub FsoExample()
    Dim fso As Object ' declare variable
    Set fso = CreateObject("Scripting.FileSystemObject") ' Set it to be a File System Object

    ' now use it to check if a file exists
    Dim myFilePath As String
    myFilePath = "C:\mypath\to\myfile.txt"
    If fso.FileExists(myFilePath) Then
        ' do something
    Else
        ' file doesn't exist
        MsgBox "File doesn't exist"
    End If
End Sub
```

Чтение текстового файла с помощью FileSystemObject

```
Const ForReading = 1
Const ForWriting = 2
Const ForAppending = 8

Sub ReadTextFileExample()
    Dim fso As Object
    Set fso = CreateObject("Scripting.FileSystemObject")

    Dim sourceFile As Object
    Dim myFilePath As String
    Dim myFileText As String

    myFilePath = "C:\mypath\to\myfile.txt"
    Set sourceFile = fso.OpenTextFile(myFilePath, ForReading)
    myFileText = sourceFile.ReadAll ' myFileText now contains the content of the text file
    sourceFile.Close ' close the file
    ' do whatever you might need to do with the text

    ' You can also read it line by line
    Dim line As String
    Set sourceFile = fso.OpenTextFile(myFilePath, ForReading)
    While Not sourceFile.AtEndOfStream ' while we are not finished reading through the file
        line = sourceFile.ReadLine
        ' do something with the line...
    Wend
    sourceFile.Close
End Sub
```

Создание текстового файла с помощью FileSystemObject

```
Sub CreateTextFileExample()
    Dim fso As Object
    Set fso = CreateObject("Scripting.FileSystemObject")

    Dim targetFile As Object
    Dim myFilePath As String
    Dim myFileText As String

    myFilePath = "C:\mypath\to\myfile.txt"
    Set targetFile = fso.CreateTextFile(myFilePath, True)  ' this will overwrite any existing
file
    targetFile.Write "This is some new text"
    targetFile.Write " And this text will appear right after the first bit of text."
    targetFile.WriteLine "This bit of text includes a newline character to ensure each write
takes its own line."
    targetFile.Close ' close the file
End Sub
```

Запись в существующий файл с помощью FileSystemObject

```
Const ForReading = 1
Const ForWriting = 2
Const ForAppending = 8


Sub WriteTextFileExample()
    Dim oFso
    Set oFso = CreateObject("Scripting.FileSystemObject")

    Dim oFile as Object
    Dim myFilePath as String
    Dim myFileText as String

    myFilePath = "C:\mypath\to\myfile.txt"
    ' First check if the file exists
    If oFso.FileExists(myFilePath) Then
        ' this will overwrite any existing filecontent with whatever you send the file
        ' to append data to the end of an existing file, use ForAppending instead
        Set oFile = oFso.OpenTextFile(myFilePath, ForWriting)
    Else
        ' create the file instead
        Set oFile = oFso.CreateTextFile(myFilePath) ' skipping the optional boolean for
overwrite if exists as we already checked that the file doesn't exist.
    End If
    oFile.Write "This is some new text"
    oFile.Write " And this text will appear right after the first bit of text."
    oFile.WriteLine "This bit of text includes a newline character to ensure each write takes
its own line."
    oFile.Close ' close the file
End Sub
```

Перечислять файлы в каталоге с помощью FileSystemObject

Ранняя привязка (требует ссылки на время выполнения сценариев Microsoft Scripting):

```
Public Sub EnumerateDirectory()
    Dim fso As Scripting.FileSystemObject
    Set fso = New Scripting.FileSystemObject

    Dim targetFolder As Folder
    Set targetFolder = fso.GetFolder("C:\")

    Dim foundFile As Variant
    For Each foundFile In targetFolder.Files
        Debug.Print foundFile.Name
    Next
End Sub
```

Поздняя оценка:

```
Public Sub EnumerateDirectory()
    Dim fso As Object
    Set fso = CreateObject("Scripting.FileSystemObject")

    Dim targetFolder As Object
    Set targetFolder = fso.GetFolder("C:\")

    Dim foundFile As Variant
    For Each foundFile In targetFolder.Files
        Debug.Print foundFile.Name
    Next
End Sub
```

Рекурсивно перечислять папки и файлы

Ранняя привязка (со ссылкой на `Microsoft Scripting Runtime`)

```
Sub EnumerateFilesAndFolders( _
    FolderPath As String, _
    Optional MaxDepth As Long = -1, _
    Optional CurrentDepth As Long = 0, _
    Optional Indentation As Long = 2)

    Dim FSO As Scripting.FileSystemObject
    Set FSO = New Scripting.FileSystemObject

    'Check the folder exists
    If FSO.FolderExists(FolderPath) Then
        Dim fldr As Scripting.Folder
        Set fldr = FSO.GetFolder(FolderPath)

        'Output the starting directory path
        If CurrentDepth = 0 Then
          Debug.Print fldr.Path
        End If

        'Enumerate the subfolders
        Dim subFldr As Scripting.Folder
        For Each subFldr In fldr.SubFolders
            Debug.Print Space$((CurrentDepth + 1) * Indentation) & subFldr.Name
            If CurrentDepth < MaxDepth Or MaxDepth = -1 Then
                'Recursively call EnumerateFilesAndFolders
```

```
                EnumerateFilesAndFolders subFldr.Path, MaxDepth, CurrentDepth + 1, Indentation
            End If
        Next subFldr

        'Enumerate the files
        Dim fil As Scripting.File
        For Each fil In fldr.Files
            Debug.Print Space$((CurrentDepth + 1) * Indentation) & fil.Name
        Next fil
    End If
End Sub
```

Вывод при вызове с такими аргументами, как: `EnumerateFilesAndFolders "C:\Test"`

```
C:\Test
  Documents
    Personal
      Budget.xls
      Recipes.doc
    Work
      Planning.doc
  Downloads
    FooBar.exe
  ReadMe.txt
```

Вывод при вызове с аргументами: `EnumerateFilesAndFolders "C:\Test", 0`

```
C:\Test
  Documents
  Downloads
  ReadMe.txt
```

Вывод при вызове с такими аргументами, как: `EnumerateFilesAndFolders "C:\Test", 1, 4`

```
C:\Test
    Documents
        Personal
        Work
    Downloads
        FooBar.exe
    ReadMe.txt
```

Расширение файла Strip из имени файла

```
Dim fso As New Scripting.FileSystemObject
Debug.Print fso.GetBaseName("MyFile.something.txt")
```

Печать `MyFile.something`

Обратите внимание, что метод `GetBaseName()` уже обрабатывает несколько периодов в имени файла.

Получить только расширение из имени файла

```
Dim fso As New Scripting.FileSystemObject
Debug.Print fso.GetExtensionName("MyFile.something.txt")
```

Печать `txt` Обратите внимание, что метод `GetExtensionName()` уже обрабатывает несколько периодов в имени файла.

Получить только путь из пути к файлу

Метод GetParentFolderName возвращает родительскую папку для любого пути. Хотя это также можно использовать с папками, возможно, более полезно для извлечения пути из абсолютного пути к файлу:

```
Dim fso As New Scripting.FileSystemObject
Debug.Print fso.GetParentFolderName("C:\Users\Me\My Documents\SomeFile.txt")
```

Печать `C:\Users\Me\My Documents`

Обратите внимание, что разделитель конечных путей не включен в возвращаемую строку.

Использование FSO.BuildPath для создания полного пути из пути к папке и имени файла

Если вы принимаете пользовательский ввод для путей к папкам, вам может потребоваться проверить обратную косую черту (`\`) перед созданием пути к файлу. Метод `FSO.BuildPath` делает это проще:

```
  Const sourceFilePath As String = "C:\Temp"  '<-- Without trailing backslash
  Const targetFilePath As String = "C:\Temp\" '<-- With trailing backslash

  Const fileName As String = "Results.txt"

  Dim FSO As FileSystemObject
  Set FSO = New FileSystemObject

  Debug.Print FSO.BuildPath(sourceFilePath, fileName)
  Debug.Print FSO.BuildPath(targetFilePath, fileName)
```

Выход:

```
C:\Temp\Results.txt
C:\Temp\Results.txt
```

Прочитайте Scripting.FileSystemObject онлайн: https://riptutorial.com/ru/vba/topic/990/scripting-filesystemobject

глава 5: Автоматизация или использование других приложений Библиотеки

Вступление

Если вы используете объекты в других приложениях как часть вашего приложения Visual Basic, вам может потребоваться установить ссылку на библиотеки объектов этих приложений. В этой документации представлен список, источники и примеры использования библиотек различных программных продуктов, таких как Windows Shell, Internet Explorer, XML HttpRequest и другие.

Синтаксис

- expression.CreateObject (Имя_объект)
- выражение; Необходимые. Выражение, возвращающее объект приложения.
- ObjectName; Требуемая строка. Имя класса создаваемого объекта. Сведения о допустимых именах классов см. В разделе «Программные идентификаторы OLE».

замечания

- MSDN-понимание автоматизации

  Когда приложение поддерживает автоматизацию, объекты, к которым может обратиться приложение, можно получить через Visual Basic. Используйте Visual Basic для управления этими объектами, вызывая методы на объекте или получая и устанавливая свойства объекта.

- MSDN-Проверка или добавление ссылки на библиотеку объектов

  Если вы используете объекты в других приложениях как часть вашего приложения Visual Basic, вам может потребоваться установить ссылку на библиотеки объектов этих приложений. Прежде чем вы сможете это сделать, сначала убедитесь, что приложение предоставляет библиотеку объектов.

- Диалоговое окно MSDN-Ссылки

  Позволяет выбрать объекты другого приложения, которые вы хотите получить в своем коде, установив ссылку на библиотеку объектов этого приложения.

- Метод MSDN-CreateObject

  Создает объект автоматизации указанного класса. Если приложение уже

запущено, CreateObject создаст новый экземпляр.

Examples

Регулярные выражения VBScript

```
Set createVBScriptRegExObject = CreateObject("vbscript.RegExp")
```

Инструменты> Референции> Регулярные выражения Microsoft VBScript #. #
Связанная DLL: VBScript.dll
Источник: Internet Explorer 1.0 и 5.5

- MSDN-Microsoft Beefs Up VBScript с регулярными выражениями
- MSDN-регулярный синтаксис выражения (скриптинг)
- experts-exchange - Использование регулярных выражений в Visual Basic для приложений и Visual Basic 6
- Как использовать регулярные выражения (регулярное выражение) в Microsoft Excel как внутри, так и в цикле на SO.
- regular-expressions.info/vbscript
- regular-expressions.info/vbscriptexample
- WIKI-Регулярное выражение

Код

Вы можете использовать эти функции для получения результатов RegEx, объединить все совпадения (если больше 1) в 1 строку и отобразить результат в ячейке excel.

```
Public Function getRegExResult(ByVal SourceString As String, Optional ByVal RegExPattern As
String = "\d+", _
    Optional ByVal isGlobalSearch As Boolean = True, Optional ByVal isCaseSensitive As Boolean
= False, Optional ByVal Delimiter As String = ";") As String

    Static RegExObject As Object
    If RegExObject Is Nothing Then
        Set RegExObject = createVBScriptRegExObject
    End If

    getRegExResult = removeLeadingDelimiter(concatObjectItems(getRegExMatches(RegExObject,
SourceString, RegExPattern, isGlobalSearch, isCaseSensitive), Delimiter), Delimiter)

End Function

Private Function getRegExMatches(ByRef RegExObj As Object, _
    ByVal SourceString As String, ByVal RegExPattern As String, ByVal isGlobalSearch As
Boolean, ByVal isCaseSensitive As Boolean) As Object

    With RegExObj
        .Global = isGlobalSearch
        .IgnoreCase = Not (isCaseSensitive) 'it is more user friendly to use positive meaning
of argument, like isCaseSensitive, than to use negative IgnoreCase
```

```
        .Pattern = RegExPattern
        Set getRegExMatches = .Execute(SourceString)
    End With

End Function

Private Function concatObjectItems(ByRef Obj As Object, Optional ByVal DelimiterCustom As
String = ";") As String
    Dim ObjElement As Variant
    For Each ObjElement In Obj
        concatObjectItems = concatObjectItems & DelimiterCustom & ObjElement.Value
    Next
End Function

Public Function removeLeadingDelimiter(ByVal SourceString As String, ByVal Delimiter As
String) As String
    If Left$(SourceString, Len(Delimiter)) = Delimiter Then
        removeLeadingDelimiter = Mid$(SourceString, Len(Delimiter) + 1)
    End If
End Function

Private Function createVBScriptRegExObject() As Object
    Set createVBScriptRegExObject = CreateObject("vbscript.RegExp") 'ex.:
createVBScriptRegExObject.Pattern
End Function
```

Скриптовый файловый системный объект

```
Set createScriptingFileSystemObject = CreateObject("Scripting.FileSystemObject")
```

Инструменты> Ссылки> Microsoft Scripting Runtime
Связанная DLL: ScrRun.dll
Источник: ОС Windows

MSDN-доступ к файлам с помощью FileSystemObject

> Модель File System Object (FSO) предоставляет объектно-ориентированный инструмент для работы с папками и файлами. Он позволяет использовать знакомый синтаксис object.method с богатым набором свойств, методов и событий для обработки папок и файлов. Вы также можете использовать традиционные инструкции и команды Visual Basic.
>
> Модель FSO дает вашему приложению возможность создавать, изменять, перемещать и удалять папки или определять, существуют ли и где существуют определенные папки. Он также позволяет вам получать информацию о папках, таких как их имена и дату их создания или последнего изменения.

Темы MSDN-FileSystemObject : « *... объясните концепцию FileSystemObject и как ее использовать* ». Exceltrick-FileSystemObject в VBA - Разъяснение Scripting.FileSystemObject

Объект словаря сценариев

```
Set dict = CreateObject("Scripting.Dictionary")
```

Инструменты> Ссылки> Microsoft Scripting Runtime
Связанная DLL: ScrRun.dll
Источник: ОС Windows

Объект Scripting.Dictionary
Объект MSDN-словаря

Объект Internet Explorer

```
Set createInternetExplorerObject = CreateObject("InternetExplorer.Application")
```

Инструменты> Референции> Microsoft Internet Controls
Связанная DLL: ieframe.dll
Источник: браузер Internet Explorer

Объект MSDN-InternetExplorer

> Управляет экземпляром Windows Internet Explorer с помощью автоматизации.

Основные пользователи Internet Explorer Objec

В приведенном ниже коде должно быть показано, как работает объект IE и как его можно манипулировать с помощью VBA. Я рекомендую пройти через него, иначе он может ошибиться во время нескольких навигаций.

```
Sub IEGetToKnow()
    Dim IE As InternetExplorer 'Reference to Microsoft Internet Controls
    Set IE = New InternetExplorer

    With IE
        .Visible = True 'Sets or gets a value that indicates whether the object is visible or
hidden.

        'Navigation
        .Navigate2 "http://www.example.com" 'Navigates the browser to a location that might
not be expressed as a URL, such as a PIDL for an entity in the Windows Shell namespace.
        Debug.Print .Busy 'Gets a value that indicates whether the object is engaged in a
navigation or downloading operation.
        Debug.Print .ReadyState 'Gets the ready state of the object.
        .Navigate2 "http://www.example.com/2"
        .GoBack 'Navigates backward one item in the history list
        .GoForward 'Navigates forward one item in the history list.
        .GoHome 'Navigates to the current home or start page.
        .Stop 'Cancels a pending navigation or download, and stops dynamic page elements, such
as background sounds and animations.
```

```
        .Refresh 'Reloads the file that is currently displayed in the object.

        Debug.Print .Silent 'Sets or gets a value that indicates whether the object can
display dialog boxes.
        Debug.Print .Type 'Gets the user type name of the contained document object.

        Debug.Print .Top 'Sets or gets the coordinate of the top edge of the object.
        Debug.Print .Left 'Sets or gets the coordinate of the left edge of the object.
        Debug.Print .Height 'Sets or gets the height of the object.
        Debug.Print .Width 'Sets or gets the width of the object.
    End With

    IE.Quit 'close the application window
End Sub
```

Веб-скребок

Наиболее распространенная проблема с IE заключается в том, чтобы очистить некоторую информацию веб-сайта или заполнить форму веб-сайта и отправить информацию. Мы рассмотрим, как это сделать.

Давайте рассмотрим исходный код *example.com* :

```
<!doctype html>
<html>
    <head>
        <title>Example Domain</title>
        <meta charset="utf-8" />
        <meta http-equiv="Content-type" content="text/html; charset=utf-8" />
        <meta name="viewport" content="width=device-width, initial-scale=1" />
        <style ... </style>
    </head>

    <body>
        <div>
            <h1>Example Domain</h1>
            <p>This domain is established to be used for illustrative examples in documents.
You may use this
            domain in examples without prior coordination or asking for permission.</p>
            <p><a href="http://www.iana.org/domains/example">More information...</a></p>
        </div>
    </body>
</html>
```

Мы можем использовать код, как показано ниже, чтобы получить и установить информацию:

```
Sub IEWebScrape1()
    Dim IE As InternetExplorer 'Reference to Microsoft Internet Controls
    Set IE = New InternetExplorer

    With IE
        .Visible = True
        .Navigate2 "http://www.example.com"
```

```
        'we add a loop to be sure the website is loaded and ready.
        'Does not work consistently. Cannot be relied upon.
        Do While .Busy = True Or .ReadyState <> READYSTATE_COMPLETE 'Equivalent = .ReadyState
<> 4
            ' DoEvents - worth considering. Know implications before you use it.
            Application.Wait (Now + TimeValue("00:00:01")) 'Wait 1 second, then check again.
        Loop

        'Print info in immediate window
        With .Document 'the source code HTML "below" the displayed page.
            Stop 'VBE Stop. Continue line by line to see what happens.
            Debug.Print .GetElementsByTagName("title")(0).innerHtml 'prints "Example Domain"
            Debug.Print .GetElementsByTagName("h1")(0).innerHtml 'prints "Example Domain"
            Debug.Print .GetElementsByTagName("p")(0).innerHtml 'prints "This domain is
established..."
            Debug.Print .GetElementsByTagName("p")(1).innerHtml 'prints "<a
href="http://www.iana.org/domains/example">More information...</a>"
            Debug.Print .GetElementsByTagName("p")(1).innerText 'prints "More information..."
            Debug.Print .GetElementsByTagName("a")(0).innerText 'prints "More information..."

            'We can change the localy displayed website. Don't worry about breaking the site.
            .GetElementsByTagName("title")(0).innerHtml = "Psst, scraping..."
            .GetElementsByTagName("h1")(0).innerHtml = "Let me try something fishy." 'You have
just changed the local HTML of the site.
            .GetElementsByTagName("p")(0).innerHtml = "Lorem ipsum.......... The End"
            .GetElementsByTagName("a")(0).innerText = "iana.org"
        End With '.document

        .Quit 'close the application window
    End With 'ie

End Sub
```

Что здесь происходит? Ключевым игроком здесь является **.Document** , то есть исходный код HTML. Мы можем применить некоторые запросы для получения Коллекций или Объекта, которые мы хотим.
Например, `IE.Document.GetElementsByTagName("title")(0).innerHtml` . `GetElementsByTagName` возвращает **коллекцию** HTML-элементов, имеющих тег « *title* ». В исходном коде есть только один тег. **Коллекция** основана на 0. Итак, чтобы получить первый элемент, добавим `(0)` . Теперь, в нашем случае, мы хотим только `innerHtml` (String), а не сам элемент элемента. Поэтому мы указываем требуемое свойство.

Нажмите

Чтобы перейти по ссылке на сайт, мы можем использовать несколько методов:

```
Sub IEGoToPlaces()
    Dim IE As InternetExplorer 'Reference to Microsoft Internet Controls
    Set IE = New InternetExplorer

    With IE
        .Visible = True
        .Navigate2 "http://www.example.com"
        Stop 'VBE Stop. Continue line by line to see what happens.
```

```
        'Click
        .Document.GetElementsByTagName("a")(0).Click
        Stop 'VBE Stop.

        'Return Back
        .GoBack
        Stop 'VBE Stop.

        'Navigate using the href attribute in the <a> tag, or "link"
        .Navigate2 .Document.GetElementsByTagName("a")(0).href
        Stop 'VBE Stop.

        .Quit 'close the application window
    End With
End Sub
```

Библиотека Microsoft HTML Object или IE Лучший друг

Чтобы получить максимальную отдачу от HTML, загружаемого в IE, вы можете (или должны) использовать другую Библиотеку, то есть *библиотеку объектов Microsoft HTML* . Подробнее об этом в другом примере.

Основные проблемы IE

Основная проблема с IE - проверка того, что страница загружена и готова к взаимодействию. `Do While... Loop` помогает, но не является надежным.

Кроме того, использование IE для очистки содержимого HTML - OVERKILL. Зачем? Поскольку браузер предназначен для просмотра, то есть отображение веб-страницы со всеми CSS, JavaScripts, картинками, всплывающими окнами и т. Д. Если вам нужны только необработанные данные, рассмотрите другой подход. Например, используя XML HTTPRequest . Подробнее об этом в другом примере.

Прочитайте Автоматизация или использование других приложений Библиотеки онлайн: https://riptutorial.com/ru/vba/topic/8916/автоматизация-или-использование-других-приложений-библиотеки

глава 6: Атрибуты

Синтаксис

- Атрибут VB_Name = "ClassOrModuleName"
- Атрибут VB_GlobalNameSpace = False 'Игнорируется
- Атрибут VB_Creatable = False 'Игнорируется
- Атрибут VB_PredeclaredId = {True | Ложь}
- Атрибут VB_Exposed = {True | Ложь}
- Атрибут variableName.VB_VarUserMemId = 0 'Zero указывает, что это член класса по умолчанию.
- Атрибут variableName.VB_VarDescription = "some string" 'Добавляет текст в информацию обозревателя объектов для этой переменной.
- Attribute procName.VB_Description = "some string" 'Добавляет текст в информацию обозревателя объектов для процедуры.
- Атрибут procName.VB_UserMemId = {0 | -4}
    - '0: Делает функцию членом по умолчанию класса.
    - '-4: Указывает, что функция возвращает Enumerator.

Examples

VB_Name

VB_Name указывает имя класса или модуля.

```
Attribute VB_Name = "Class1"
```

Новый экземпляр этого класса будет создан с помощью

```
Dim myClass As Class1
myClass = new Class1
```

VB_GlobalNameSpace

В VBA этот атрибут игнорируется. Он не был перенесен с VB6.

В VB6 он создает глобальный экземпляр по умолчанию класса («ярлык»), так что к членам класса можно получить доступ без использования имени класса. Например, `DateTime` (как в `DateTime.Now`) на самом деле является частью класса `VBA.Conversion` .

```
Debug.Print VBA.Conversion.DateTime.Now
Debug.Print DateTime.Now
```

VB_Createable

Этот атрибут игнорируется. Он не был перенесен с VB6.

В VB6 он использовался в сочетании с атрибутом `VB_Exposed` для контроля доступности классов за пределами текущего проекта.

```
VB_Exposed=True
VB_Creatable=True
```

Это приведет к `Public Class` , к которому можно получить доступ из других проектов, но эта функциональность не существует в VBA.

VB_PredeclaredId

Создает глобальный экземпляр по умолчанию для класса. Экземпляр по умолчанию получает доступ через имя класса.

декларация

```
VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "Class1"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False
Option Explicit

Public Function GiveMeATwo() As Integer
    GiveMeATwo = 2
End Function
```

Вызов

```
Debug.Print Class1.GiveMeATwo
```

В некотором смысле это моделирует поведение статических классов на других языках, но в отличие от других языков вы все равно можете создать экземпляр класса.

```
Dim cls As Class1
Set cls = New Class1
Debug.Print cls.GiveMeATwo
```

VB_Exposed

Управляет инстинктивными характеристиками класса.

```
Attribute VB_Exposed = False
```

Делает класс `Private` . Он не может быть доступен за пределами текущего проекта.

```
Attribute VB_Exposed = True
```

Выдает класс `Public` , вне проекта. Однако, поскольку `VB_Createable` игнорируется в VBA, экземпляры класса не могут быть созданы напрямую. Это эквивалентно следующему классу VB.Net.

```
Public Class Foo
    Friend Sub New()
    End Sub
End Class
```

Чтобы получить экземпляр извне проекта, вы должны открыть фабрику для создания экземпляров. Один из способов сделать это - с помощью обычного `Public` модуля.

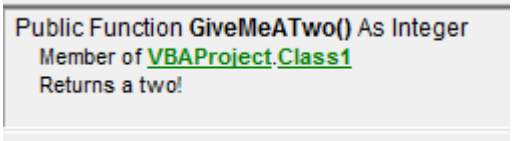
```
Public Function CreateFoo() As Foo
    CreateFoo = New Foo
End Function
```

Поскольку общедоступные модули доступны из других проектов, это позволяет нам создавать новые экземпляры наших классов `Public - Not Createable` .

VB_Description

Добавляет текстовое описание к члену класса или модуля, который становится видимым в Проводнике объектов. В идеале все публичные члены публичного интерфейса / API должны иметь описание.

```
Public Function GiveMeATwo() As Integer
    Attribute GiveMeATwo.VB_Description = "Returns a two!"
    GiveMeATwo = 2
End Property
```



Примечание. Все элементы доступа к объекту (`Get` , `Let` , `Set`) используют одно и то же описание.

VB_ [вар] UserMemId

`VB_VarUserMemId` (для переменных области модуля) и `VB_UserMemId` (для процедур) используются в VBA в основном для двух вещей.

Указание члена класса по умолчанию для класса

Класс `List` который будет инкапсулировать `Collection` должен иметь свойство `Item` , поэтому код клиента может сделать это:

```
For i = 1 To myList.Count 'VBA Collection Objects are 1-based
    Debug.Print myList.Item(i)
Next
```

Но с атрибутом `VB_UserMemId` установленным на 0 в свойстве `Item` , код клиента может сделать это:

```
For i = 1 To myList.Count 'VBA Collection Objects are 1-based
    Debug.Print myList(i)
Next
```

Только один член может юридически иметь `VB_UserMemId = 0` в любом заданном классе. Для свойств укажите атрибут в `Get` Access:

```
Option Explicit
Private internal As New Collection

Public Property Get Count() As Long
    Count = internal.Count
End Property

Public Property Get Item(ByVal index As Long) As Variant
Attribute Item.VB_Description = "Gets or sets the element at the specified index."
Attribute Item.VB_UserMemId = 0
'Gets the element at the specified index.
    Item = internal(index)
End Property

Public Property Let Item(ByVal index As Long, ByVal value As Variant)
'Sets the element at the specified index.
    With internal
        If index = .Count + 1 Then
            .Add item:=value
        ElseIf index = .Count Then
            .Remove index
            .Add item:=value
        ElseIf index < .Count Then
            .Remove index
            .Add item:=value, before:=index
        End If
    End With
End Property
```

Создание класса с итерацией с помощью конструкции `For Each` loop

С магическим значением `-4` , атрибут `VB_UserMemId` сообщает VBA, что этот член дает перечислитель, который позволяет клиенту сделать это:

```
Dim item As Variant
For Each item In myList
    Debug.Print item
Next
```

Самый простой способ реализовать этот метод - вызвать скрытый `[_NewEnum]` свойства `[_NewEnum]` во внутренней / инкапсулированной `Collection` ; идентификатор должен быть заключен в квадратные скобки из-за ведущего подчеркивания, что делает его незаконным идентификатором VBA:

```
Public Property Get NewEnum() As IUnknown
Attribute NewEnum.VB_Description = "Gets an enumerator that iterates through the List."
Attribute NewEnum.VB_UserMemId = -4
Attribute NewEnum.VB_MemberFlags = "40" 'would hide the member in VB6. not supported in VBA.
'Gets an enumerator that iterates through the List.
    Set NewEnum = internal.[_NewEnum]
End Property
```

Прочитайте Атрибуты онлайн: https://riptutorial.com/ru/vba/topic/5321/атрибуты

глава 7: Измерение длины строк

замечания

Длина строки может быть измерена двумя способами: наиболее часто используемым измерением длины является количество символов, использующих функции `Len` , но VBA также может показывать количество байтов с использованием функций `LenB` . Двухбайтовый или Unicode-символ имеет длину более одного байта.

Examples

Используйте функцию Len для определения количества символов в строке

```
Const baseString As String = "Hello World"

Dim charLength As Long

charLength = Len(baseString)
'charlength = 11
```

Используйте функцию LenB для определения количества байтов в строке

```
Const baseString As String = "Hello World"

Dim byteLength As Long

byteLength = LenB(baseString)
'byteLength = 22
```

Предпочитаю `Если Len (myString) = 0 Затем` over `If myString =" "Then`

При проверке, является ли строка нулевой длиной, лучше использовать и более эффективно проверять длину строки, а не сравнивать строку с пустой строкой.

```
Const myString As String = vbNullString

'Prefer this method when checking if myString is a zero-length string
If Len(myString) = 0 Then
    Debug.Print "myString is zero-length"
End If

'Avoid using this method when checking if myString is a zero-length string
If myString = vbNullString Then
    Debug.Print "myString is zero-length"
End If
```

Прочитайте Измерение длины строк онлайн: https://riptutorial.com/ru/vba/topic/3576/измерение-длины-строк

глава 8: Интерфейсы

Вступление

Интерфейс - это способ определить набор действий, которые будет выполнять класс. Определение интерфейса - это список сигнатур методов (имя, параметры и тип возврата). Говорят, что класс, имеющий все методы, «реализует» этот интерфейс.

В VBA использование интерфейсов позволяет компилятору проверить, что модуль реализует все его методы. Переменная или параметр могут быть определены в терминах интерфейса вместо определенного класса.

Examples

Простой интерфейс - доступный

Интерфейс `Flyable` - это модуль класса со следующим кодом:

```
Public Sub Fly()
    ' No code.
End Sub

Public Function GetAltitude() As Long
    ' No code.
End Function
```

Модуль класса, `Airplane` , использует `Implements` ключевое слово , чтобы сообщить компилятору сгенерировать ошибку , если она не имеет два метода: `Flyable_Fly()` к югу и `Flyable_GetAltitude()` функция , которая возвращает `Long` .

```
Implements Flyable

Public Sub Flyable_Fly()
    Debug.Print "Flying With Jet Engines!"
End Sub

Public Function Flyable_GetAltitude() As Long
    Flyable_GetAltitude = 10000
End Function
```

Модуль второго класса, `Duck` , также реализует `Flyable` :

```
Implements Flyable

Public Sub Flyable_Fly()
    Debug.Print "Flying With Wings!"
End Sub
```

```
Public Function Flyable_GetAltitude() As Long
    Flyable_GetAltitude = 30
End Function
```

Мы можем написать процедуру, которая принимает любое значение `Flyable` , зная, что она ответит команде `Fly` или `GetAltitude` :

```
Public Sub FlyAndCheckAltitude(F As Flyable)
    F.Fly
    Debug.Print F.GetAltitude
End Sub
```

Поскольку интерфейс определен, всплывающее окно IntelliSense покажет `Fly` и `GetAltitude` для `F`

Когда мы запускаем следующий код:

```
Dim MyDuck As New Duck
Dim MyAirplane As New Airplane

FlyAndCheckAltitude MyDuck
FlyAndCheckAltitude MyAirplane
```

Выход:

```
Flying With Wings!
30
Flying With Jet Engines!
10000
```

Обратите внимание, что, хотя подпрограмма называется `Flyable_Fly` как на `Airplane` и на `Duck` , ее можно назвать « `Fly` когда переменная или параметр определяется как « `Flyable` . Если переменная определена как `Duck` , ее нужно будет назвать `Flyable_Fly` .

Несколько интерфейсов в одном классе - плавающие и плавающие

Используя пример `Flyable` в качестве отправной точки, мы можем добавить второй интерфейс `Swimmable` со следующим кодом:

```
Sub Swim()
    ' No code
End Sub
```

Объект `Duck` может `Implement` как летающие, так и плавающие:

```
Implements Flyable
Implements Swimmable

Public Sub Flyable_Fly()
    Debug.Print "Flying With Wings!"
```

```
End Sub

Public Function Flyable_GetAltitude() As Long
    Flyable_GetAltitude = 30
End Function

Public Sub Swimmable_Swim()
    Debug.Print "Floating on the water"
End Sub
```

Класс `Fish` может реализовать `Swimmable` :

```
Implements Swimmable

Public Sub Swimmable_Swim()
    Debug.Print "Swimming under the water"
End Sub
```

Теперь мы видим, что объект `Duck` может быть передан Sub в качестве `Flyable` с одной стороны, а `Swimmable` с другой:

```
Sub InterfaceTest()

Dim MyDuck As New Duck
Dim MyAirplane As New Airplane
Dim MyFish As New Fish

Debug.Print "Fly Check..."

FlyAndCheckAltitude MyDuck
FlyAndCheckAltitude MyAirplane

Debug.Print "Swim Check..."

TrySwimming MyDuck
TrySwimming MyFish

End Sub

Public Sub FlyAndCheckAltitude(F As Flyable)
    F.Fly
    Debug.Print F.GetAltitude
End Sub

Public Sub TrySwimming(S As Swimmable)
    S.Swim
End Sub
```

Результат этого кода:

> Fly Check ...
>
> Полет с крыльями!
>
> 30

Полет с реактивными двигателями!

10000

Плавать проверить ...

Плавающий на воде

Плавание под водой

Прочитайте Интерфейсы онлайн: https://riptutorial.com/ru/vba/topic/8784/интерфейсы

глава 9: Ключевое слово Option VBA

Синтаксис

- Опция optionName [значение]
- Вариант Явный
- Вариант сравнения {Текст | Двоичные | База данных}
- Дополнительный модуль
- База опций {0 | 1}

параметры

вариант	подробность
Явный	*Требовать объявление переменной* в указанном модуле (в идеале, все); с указанной опцией, использование необъявленной (/ неверно введенной) переменной становится ошибкой компиляции.
Сравнить текст	Делает сравнения строк в модуле нечувствительными к регистру на основе языкового стандарта системы, приоритетом которого является алфавитная эквивалентность (например, «a» = «A»).
Сравнить двоичный	Режим сравнения строк по умолчанию. Делает сравнения строк в модуле чувствительными к регистру, сравнивая строки, используя двоичное представление / числовое значение каждого символа (например, ASCII).
Сравнить базы данных	(Только для MS-Access) Делает сравнения строк в модуле так, как в SQL-заявлении.
Частный модуль	Препятствует модуля `Public` члена от доступа извне проекта , что модуль проживает в, эффективно скрывая процедуры от хост - приложения (то есть , не доступные для использования в качестве макросов или функций , определяемых пользователем).
База опций 0	Настройки по умолчанию. Устанавливает ограничение неявного массива на `0` в модуле. Когда массив объявляется без явного нижнего граничного значения `0` будет использоваться.
Вариант базы 1	Устанавливает ограничение неявного массива на `1` в модуле. Когда массив объявляется без явного нижнего граничного значения `1` будет использоваться.

замечания

Намного проще управлять границами массивов, явно объявляя границы, а не позволяя компилятору вернуться к объявлению `Option Base {0|1}` . Это можно сделать так:

```
Dim myStringsA(0 To 5) As String '// This has 6 elements (0 - 5)
Dim myStringsB(1 To 5) As String '// This has 5 elements (1 - 5)
Dim myStringsC(6 To 9) As String '// This has 3 elements (6 - 9)
```

Examples

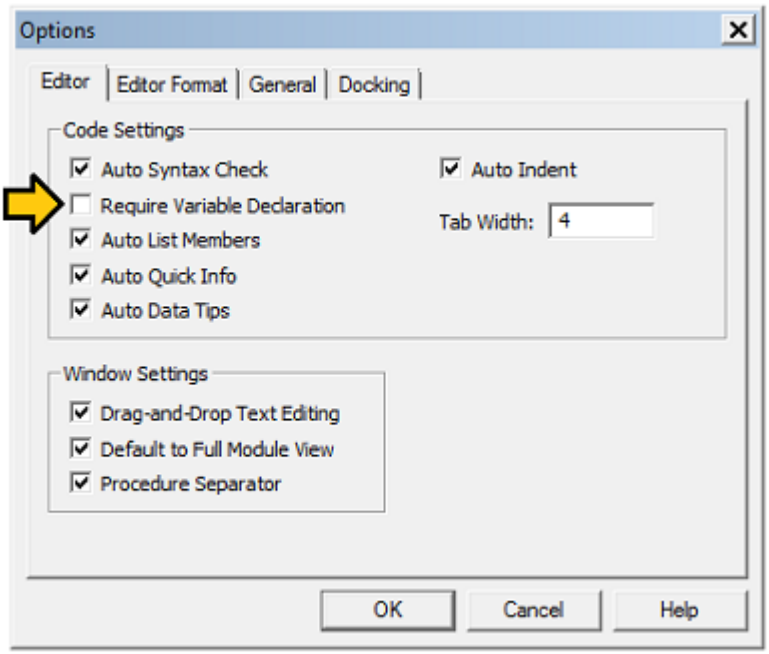
Вариант Явный

Рекомендуется использовать `Option Explicit` в VBA, поскольку он заставляет разработчика объявлять все свои переменные перед использованием. Это также имеет другие преимущества, такие как автоматическая капитализация объявленных имен переменных и IntelliSense.

```
Option Explicit

Sub OptionExplicit()
    Dim a As Integer
    a = 5
    b = 10 '// Causes compile error as 'b' is not declared
End Sub
```

Настройка **Требовать объявление переменной** в инструментах VBE ► `Функции` ► `Страница свойств редактора помещает оператор` **Option Explicit** в начало каждого вновь созданного листа кода.



Это позволит избежать ошибок глупого кодирования, таких как орфографические ошибки, а также повлиять на использование правильного типа переменной в объявлении переменной. (Еще несколько примеров приведены ВСЕГДА Используйте «Option Explicit» .)

Вариант сравнения {Binary | Текст | База данных}

Вариант сравнения Двоичный

Двоичное сравнение делает все проверки для равенства строк внутри *чувствительного* к модулю / классу. Технически с этой опцией строковые сравнения выполняются с использованием порядка сортировки двоичных представлений каждого символа.

    A <B <E <Z <a <b <e <z

Если в модуле не задано значение Option Compare, Binary используется по умолчанию.

```
Option Compare Binary

Sub CompareBinary()

    Dim foo As String
    Dim bar As String

    '// Case sensitive
    foo = "abc"
    bar = "ABC"

    Debug.Print (foo = bar) '// Prints "False"

    '// Still differentiates accented characters
    foo = "ábc"
    bar = "abc"

    Debug.Print (foo = bar) '// Prints "False"

    '// "b" (Chr 98) is greater than "a" (Chr 97)
    foo = "a"
    bar = "b"

    Debug.Print (bar > foo) '// Prints "True"

    '// "b" (Chr 98) is NOT greater than "á" (Chr 225)
    foo = "á"
    bar = "b"

    Debug.Print (bar > foo) '// Prints "False"

End Sub
```

Опция Сравнить текст

Параметр Compare Text позволяет сравнивать все строки в модуле / классе с

использованием *нечувствительного к регистру* сравнения.

(A | a) <(B | b) <(Z | z)

```
Option Compare Text

Sub CompareText()

    Dim foo As String
    Dim bar As String

    '// Case insensitivity
    foo = "abc"
    bar = "ABC"

    Debug.Print (foo = bar) '// Prints "True"

    '// Still differentiates accented characters
    foo = "ábc"
    bar = "abc"

    Debug.Print (foo = bar) '// Prints "False"

    '// "b" still comes after "a" or "á"
    foo = "á"
    bar = "b"

    Debug.Print (bar > foo) '// Prints "True"

End Sub
```

База данных сравнения опций

База данных сравнения опций доступна только в MS Access. Он устанавливает модуль / класс для использования текущих параметров базы данных, чтобы определить, следует ли использовать текстовый или двоичный режим.

Примечание. Использование этого параметра не рекомендуется, если только модуль не используется для записи пользовательских UDF доступа (функций, определенных пользователем), которые должны обрабатывать сопоставления текста так же, как SQL-запросы в этой базе данных.

База опций {0 | 1}

`Option Base` используется для объявления нижней границы элементов **массива** по умолчанию. Он объявляется на уровне модуля и действителен только для текущего модуля.

По умолчанию (и, следовательно, если база опций не указана), Base равно 0. Это означает, что первый элемент любого массива, объявленного в модуле, имеет индекс 0.

Если задана `Option Base 1` , первый элемент массива имеет индекс 1

Пример в базе 0:

```
Option Base 0

Sub BaseZero()

    Dim myStrings As Variant

    ' Create an array out of the Variant, having 3 fruits elements
    myStrings = Array("Apple", "Orange", "Peach")

    Debug.Print LBound(myStrings) ' This Prints "0"
    Debug.Print UBound(myStrings) ' This print "2", because we have 3 elements beginning at 0
-> 0,1,2

    For i = 0 To UBound(myStrings)

        Debug.Print myStrings(i) ' This will print "Apple", then "Orange", then "Peach"

    Next i

End Sub
```

Тот же пример с базой 1

```
Option Base 1

Sub BaseOne()

    Dim myStrings As Variant

    ' Create an array out of the Variant, having 3 fruits elements
    myStrings = Array("Apple", "Orange", "Peach")

    Debug.Print LBound(myStrings) ' This Prints "1"
    Debug.Print UBound(myStrings) ' This print "3", because we have 3 elements beginning at 1
-> 1,2,3

    For i = 0 To UBound(myStrings)

        Debug.Print myStrings(i) ' This triggers an error 9 "Subscript out of range"

    Next i

End Sub
```

Второй пример сгенерировал подстрочный индекс (Error 9) на первом этапе цикла, поскольку была предпринята попытка получить доступ к индексу 0 массива, и этот индекс не существует, поскольку модуль объявлен с `Base 1`

Правильный код с базой 1:

```
    For i = 1 To UBound(myStrings)
```

```
        Debug.Print myStrings(i) ' This will print "Apple", then "Orange", then "Peach"
    Next i
```

Следует отметить, что функция Split **всегда** создает массив с индексом элемента на основе нуля независимо от любого параметра `Option Base` . Примеры использования функции **Split** можно найти здесь.

> Функция разделения
>
> > Возвращает одномерный массив на основе нуля, содержащий указанное количество подстрок.

В Excel свойства `Range.Value` и `Range.Formula` для диапазона с несколькими `Range.Formula` *всегда* возвращают массив 2D `Range.Formula` на основе 1.

Аналогично, в ADO метод `Recordset.GetRows` *всегда* возвращает 2D-массив на основе 1.

Одна из рекомендуемых «лучших практик» - всегда использовать функции LBound и UBound для определения экстентов массива.

```
'for single dimensioned array
Debug.Print LBound(arr) & ":" & UBound(arr)
Dim i As Long
For i = LBound(arr) To UBound(arr)
    Debug.Print arr(i)
Next i

'for two dimensioned array
Debug.Print LBound(arr, 1) & ":" & UBound(arr, 1)
Debug.Print LBound(arr, 2) & ":" & UBound(arr, 2)
Dim i As long, j As Long
For i = LBound(arr, 1) To UBound(arr, 1)
    For j = LBound(arr, 2) To UBound(arr, 2)
         Debug.Print arr(i, j)
    Next j
Next i
```

База `Option Base 1` должна находиться в верхней части каждого модуля кода, где массив создается или изменен, если массивы должны последовательно создаваться с нижней границей 1.

Прочитайте Ключевое слово Option VBA онлайн: https://riptutorial.com/ru/vba/topic/3992/ключевое-слово-option-vba

глава 10: Коллекции

замечания

`Collection` представляет собой контейнерный объект, который включен в среду выполнения VBA. Никаких дополнительных ссылок для его использования не требуется. `Collection` может использоваться для хранения элементов любого типа данных и позволяет извлекать либо порядковый индекс элемента, либо с помощью необязательного уникального ключа.

Сравнение функций с массивами и словарями

	Коллекция	массив	толковый словарь
Могут быть изменены	да	Иногда [1]	да
Элементы заказаны	да	да	Да [2]
Элементы строго типизированы	нет	да	нет
Элементы могут быть получены по порядковым номерам	да	да	нет
Новые элементы могут быть вставлены по порядковому номеру	да	нет	нет
Как определить, существует ли элемент	Итерировать все элементы	Итерировать все элементы	Итерировать все элементы
Элементы могут быть получены ключом	да	нет	да
Ключи чувствительны к регистру	нет	N / A	Дополнительно [3]
Как определить, существует ли ключ	Обработчик ошибок	N / A	`.Exists` функция
Удалить все элементы	Итерация и	`Erase` , `ReDim`	`.RemoveAll`

	Коллекция	массив	толковый словарь
	`.Remove`		функция

[1] Могут быть изменены только динамические массивы и только последнее измерение многомерных массивов.

[2] , лежащие в основе `.Keys` и `.Items` упорядочены.

[3] Определяется свойством `.CompareMode` .

Examples

Добавление элементов в коллекцию

Элементы добавляются в `Collection` , вызывая метод `.Add` :

Синтаксис:

```
.Add(item, [key], [before, after])
```

параметр	Описание
вещь	Элемент для хранения в `Collection` . Это может быть практически любое значение, которому может быть присвоена переменная, включая примитивные типы, массивы, объекты и `Nothing` .
ключ	Необязательный. `String` которая служит уникальным идентификатором для извлечения элементов из `Collection` . Если указанный ключ уже существует в `Collection` , это приведет к ошибке времени выполнения 457: «Этот ключ уже связан с элементом этой коллекции».
до	Необязательный. Существующий ключ (`String` value) *или* индекс (числовое значение) для вставки элемента в `Collection` . Если задано значение, параметр *after* ***должен*** быть пустым или ошибка времени выполнения 5: «Неверный вызов или аргумент процедуры». Если передан `String` ключ, который не существует в `Collection` , это приведет к ошибке времени выполнения 5: «Неверный вызов или аргумент процедуры». Если числовой индекс передан, который не существует в `Collection` , это приведет к ошибке времени выполнения 9: «Подзаголовок вне диапазона».
после	Необязательный. Существующий ключ (`String` value) *или* индекс (числовое значение) для вставки элемента после в `Collection` . Если задано значение, параметр *before* ***должен*** быть пустым. Исправленные ошибки идентичны

параметр	Описание
	параметру *before* .

Заметки:

- Ключи ***не*** чувствительны к регистру. `.Add "Bar", "Foo"` и `.Add "Baz", "foo"` приведет к столкновению клавиш.
- Если ни один из необязательных параметров *до* или *после* не задан, элемент будет добавлен после последнего элемента `Collection` .
- Вставки, заданные параметром *до* или *после* , изменят числовые индексы существующих элементов, чтобы они соответствовали новой позиции. Это означает, что при вставках в циклы следует соблюдать осторожность, используя числовые индексы.

Пример использования:

```
Public Sub Example()
    Dim foo As New Collection

    With foo
        .Add "One"            'No key. This item can only be retrieved by index.
        .Add "Two", "Second"  'Key given. Can be retrieved by key or index.
        .Add "Three", , 1     'Inserted at the start of the collection.
        .Add "Four", , , 1    'Inserted at index 2.
    End With

    Dim member As Variant
    For Each member In foo
        Debug.Print member    'Prints "Three, Four, One, Two"
    Next
End Sub
```

Удаление элементов из коллекции

Элементы удаляются из `Collection` , вызывая ее метод `.Remove` :

Синтаксис:

```
.Remove(index)
```

параметр	Описание
индекс	Элемент для удаления из `Collection` . Если переданное значение является числовым или `Variant` с числовым подтипом, оно будет интерпретироваться как числовой индекс. Если переданное значение представляет собой

параметр	Описание
	`String` или `Variant` содержащий строку, это будет интерпретироваться как ключ a. Если передан String-ключ, который не существует в `Collection` , это приведет к ошибке времени выполнения 5: «Неверный вызов или аргумент процедуры». Если числовой индекс передан, который не существует в `Collection` , это приведет к ошибке времени выполнения 9: «Подзаголовок вне диапазона».

Заметки:

- Удаление элемента из `Collection` изменит числовые индексы всех элементов после него в `Collection` . `For` циклов, которые используют числовые индексы и удаляющие элементы, должны работать в *обратном направлении* (`Step -1`), чтобы исключить исключения в индексе и пропущенные элементы.
- Элементы обычно ***не*** должны удаляться из `Collection` изнутри цикла `For Each` поскольку это может дать непредсказуемые результаты.

Пример использования:

```
Public Sub Example()
    Dim foo As New Collection

    With foo
        .Add "One"
        .Add "Two", "Second"
        .Add "Three"
        .Add "Four"
    End With

    foo.Remove 1            'Removes the first item.
    foo.Remove "Second"     'Removes the item with key "Second".
    foo.Remove foo.Count    'Removes the last item.

    Dim member As Variant
    For Each member In foo
        Debug.Print member  'Prints "Three"
    Next
End Sub
```

Получение количества предметов коллекции

Количество элементов в `Collection` можно получить, вызвав ее функцию `.Count` :

Синтаксис:

```
.Count()
```

Пример использования:

```
Public Sub Example()
    Dim foo As New Collection

    With foo
        .Add "One"
        .Add "Two"
        .Add "Three"
        .Add "Four"
    End With

    Debug.Print foo.Count   'Prints 4
End Sub
```

Извлечение предметов из коллекции

Элементы можно получить из `Collection` , вызвав функцию `.Item` .

Синтаксис:

```
.Item(index)
```

параметр	Описание
индекс	Элемент для извлечения из `Collection` . Если переданное значение является числовым или `Variant` с числовым подтипом, оно будет интерпретироваться как числовой индекс. Если переданное значение представляет собой `String` или `Variant` содержащий строку, это будет интерпретироваться как ключ a. Если передан String-ключ, который не существует в `Collection` , это приведет к ошибке времени выполнения 5: «Неверный вызов или аргумент процедуры». Если числовой индекс передан, который не существует в `Collection` , это приведет к ошибке времени выполнения 9: «Подзаголовок вне диапазона».

Заметки:

- `.Item` является членом `Collection` по умолчанию. Это обеспечивает гибкость в синтаксисе, как показано в примере использования ниже.
- Числовые индексы основаны на 1.
- Ключи ***не*** чувствительны к регистру. `.Item("Foo")` и `.Item("foo")` относятся к одному и тому же ключу.
- Параметр *индекс* ***не*** неявно приводится к числу из `String` или визы-Versa. Вполне возможно, что `.Item(1)` и `.Item("1")` относятся к различным элементам `Collection` .

Пример использования (индексы):

```
Public Sub Example()
    Dim foo As New Collection

    With foo
        .Add "One"
        .Add "Two"
        .Add "Three"
        .Add "Four"
    End With

    Dim index As Long
    For index = 1 To foo.Count
        Debug.Print foo.Item(index) 'Prints One, Two, Three, Four
    Next
End Sub
```

Пример использования (ключи):

```
Public Sub Example()
    Dim keys() As String
    keys = Split("Foo,Bar,Baz", ",")
    Dim values() As String
    values = Split("One,Two,Three", ",")

    Dim foo As New Collection
    Dim index As Long
    For index = LBound(values) To UBound(values)
        foo.Add values(index), keys(index)
    Next

    Debug.Print foo.Item("Bar") 'Prints "Two"
End Sub
```

Пример использования (альтернативный синтаксис):

```
Public Sub Example()
    Dim foo As New Collection

    With foo
        .Add "One", "Foo"
        .Add "Two", "Bar"
        .Add "Three", "Baz"
    End With

    'All lines below print "Two"
    Debug.Print foo.Item("Bar")     'Explicit call syntax.
    Debug.Print foo("Bar")          'Default member call syntax.
    Debug.Print foo!Bar             'Bang syntax.
End Sub
```

Обратите внимание, что синтаксис bang (`!`) Разрешен, потому что `.Item` является членом по умолчанию и может принимать один аргумент `String` . Утилита этого синтаксиса сомнительна.

Ключи

В отличие от Scripting.Dictionary , в `Collection` нет способа определить, существует ли данный ключ *или* способ получить ключи, которые присутствуют в `Collection` . Единственный способ определить, присутствует ли ключ, - использовать обработчик ошибок:

```
Public Function KeyExistsInCollection(ByVal key As String, _
                                      ByRef container As Collection) As Boolean
    With Err
        If container Is Nothing Then .Raise 91
        On Error Resume Next
        Dim temp As Variant
        temp = container.Item(key)
        On Error GoTo 0

        If .Number = 0 Then
            KeyExistsInCollection = True
        ElseIf .Number <> 5 Then
            .Raise .Number
        End If
    End With
End Function
```

Предметы

Единственный способ определить, содержится ли элемент в `Collection` - это перебирать `Collection` до тех пор, пока элемент не будет расположен. Обратите внимание, что поскольку `Collection` может содержать как примитивы, так и объекты, требуется дополнительная обработка, чтобы избежать ошибок во время сравнений:

```
Public Function ItemExistsInCollection(ByRef target As Variant, _
                                       ByRef container As Collection) As Boolean
    Dim candidate As Variant
    Dim found As Boolean

    For Each candidate In container
        Select Case True
            Case IsObject(candidate) And IsObject(target)
                found = candidate Is target
            Case IsObject(candidate), IsObject(target)
                found = False
            Case Else
                found = (candidate = target)
        End Select
        If found Then
            ItemExistsInCollection = True
            Exit Function
        End If
```

```
    Next
End Function
```

Очистка всех элементов из коллекции

Самый простой способ очистить все элементы из `Collection` - просто заменить его на новую `Collection` а старая - выйти из области видимости:

```
Public Sub Example()
    Dim foo As New Collection

    With foo
        .Add "One"
        .Add "Two"
        .Add "Three"
    End With

    Debug.Print foo.Count   'Prints 3
    Set foo = New Collection
    Debug.Print foo.Count   'Prints 0
End Sub
```

Однако, если имеется несколько ссылок на `Collection` , этот метод даст вам только пустую `Collection` *для назначенной переменной* .

```
Public Sub Example()
    Dim foo As New Collection
    Dim bar As Collection

    With foo
        .Add "One"
        .Add "Two"
        .Add "Three"
    End With

    Set bar = foo
    Set foo = New Collection

    Debug.Print foo.Count   'Prints 0
    Debug.Print bar.Count   'Prints 3
End Sub
```

В этом случае самый простой способ очистить содержимое - это перебрать количество элементов в `Collection` и повторно удалить самый нижний элемент:

```
Public Sub ClearCollection(ByRef container As Collection)
    Dim index As Long
    For index = 1 To container.Count
        container.Remove 1
    Next
End Sub
```

Прочитайте Коллекции онлайн: https://riptutorial.com/ru/vba/topic/5838/коллекции

глава 11: Комментарии

замечания

Блоки комментариев

Если вам нужно прокомментировать или раскомментировать несколько строк одновременно, вы можете использовать кнопки **панели инструментов** « **Редактирование**» IDE:

Блок комментариев - добавляет один апостроф к началу всех выбранных строк

Uncomment Block - удаляет первый апостроф с начала всех выбранных строк

Многострочные комментарии Многие другие языки поддерживают многострочные комментарии блоков, но VBA допускает только однострочные комментарии.

Examples

Апостроф Комментарии

Комментарий отмечен апострофом (') и игнорируется при выполнении кода. Комментарии помогают объяснить ваш код будущим читателям, включая вас самих.

Поскольку все строки, начинающиеся с комментария, игнорируются, их также можно использовать для предотвращения выполнения кода (при отладке или рефакторе). Помещение апострофа ' перед тем, как ваш код превратит его в комментарий. (Это называется *комментированием* строки.)

```
Sub InlineDocumentation()
  'Comments start with an "'"

  'They can be place before a line of code, which prevents the line from executing
  'Debug.Print "Hello World"

  'They can also be placed after a statement
  'The statement still executes, until the compiler arrives at the comment
  Debug.Print "Hello World"  'Prints a welcome message

'Comments can have 0 indention....
    '... or as much as needed

  '''' Comments can contain multiple apostrophes ''''
```

```
  'Comments can span lines (using line continuations) _
    but this can make for hard to read code

  'If you need to have mult-line comments, it is often easier to
  'use an apostrophe on each line

  'The continued statement syntax (:) is treated as part of the comment, so
  'it is not possible to place an executable statement after a comment
  'This won't run : Debug.Print "Hello World"
End Sub

'Comments can appear inside or outside a procedure
```

Комментарии REM

```
Sub RemComments()
  Rem Comments start with "Rem" (VBA will change any alternate casing to "Rem")
  Rem is an abbreviation of Remark, and similar to DOS syntax
  Rem Is a legacy approach to adding comments, and apostrophes should be preferred

  Rem Comments CANNOT appear after a statement, use the apostrophe syntax instead
  Rem Unless they are preceded by the instruction separator token
  Debug.Print "Hello World": Rem prints a welcome message
  Debug.Print "Hello World" 'Prints a welcome message

  'Rem cannot be immediately followed by the following characters "!,@,#,$,%,&"
  'Whereas the apostrophe syntax can be followed by any printable character.

End Sub

Rem Comments can appear inside or outside a procedure
```

Прочитайте Комментарии онлайн: https://riptutorial.com/ru/vba/topic/2059/комментарии

глава 12: Конкатенация строк

замечания

Строки могут быть объединены или соединены друг с другом, используя один или более оператор конкатенации `&` .

Строковые массивы также могут быть объединены с использованием функции `Join` и предоставления строки (которая может быть нулевой длины), которая будет использоваться между каждым элементом массива.

Examples

Конкатенация строк с помощью оператора &

```
Const string1 As String = "foo"
Const string2 As String = "bar"
Const string3 As String = "fizz"
Dim concatenatedString As String

'Concatenate two strings
concatenatedString = string1 & string2
'concatenatedString = "foobar"

'Concatenate three strings
concatenatedString = string1 & string2 & string3
'concatenatedString = "foobarfizz"
```

Объединение массива строк с использованием функции Join

```
'Declare and assign a string array
Dim widgetNames(2) As String
widgetNames(0) = "foo"
widgetNames(1) = "bar"
widgetNames(2) = "fizz"

'Concatenate with Join and separate each element with a 3-character string
concatenatedString = VBA.Strings.Join(widgetNames, " > ")
'concatenatedString = "foo > bar > fizz"

'Concatenate with Join and separate each element with a zero-width string
concatenatedString = VBA.Strings.Join(widgetNames, vbNullString)
'concatenatedString = "foobarfizz"
```

Прочитайте Конкатенация строк онлайн: https://riptutorial.com/ru/vba/topic/3580/конкатенация-строк

глава 13: Копирование, возврат и передача массивов

Examples

Копирование массивов

Вы можете скопировать массив VBA в массив того же типа, используя оператор `=` . Массивы должны быть одного типа, иначе код будет генерировать ошибку компиляции « Can not assign to array».

```
Dim source(0 to 2) As Long
Dim destinationLong() As Long
Dim destinationDouble() As Double

destinationLong = source     ' copies contents of source into destinationLong
destinationDouble = source   ' does not compile
```

Исходный массив может быть фиксированным или динамическим, но целевой массив должен быть динамическим. Попытка скопировать в фиксированный массив вызовет ошибку компиляции «Can not assign to array». Любые существующие данные в принимающем массиве теряются, а его границы и размеры изменяются так же, как исходный массив.

```
Dim source() As Long
ReDim source(0 To 2)

Dim fixed(0 To 2) As Long
Dim dynamic() As Long

fixed = source   ' does not compile
dynamic = source ' does compile

Dim dynamic2() As Long
ReDim dynamic2(0 to 6, 3 to 99)

dynamic2 = source ' dynamic2 now has dimension (0 to 2)
```

Как только копия сделана, два массива являются отдельными в памяти, то есть две переменные не являются ссылками на одни и те же базовые данные, поэтому изменения, внесенные в один массив, не отображаются в другом.

```
Dim source(0 To 2) As Long
Dim destination() As Long

source(0) = 3
source(1) = 1
source(2) = 4
```

```
destination = source
destination(0) = 2

Debug.Print source(0); source(1); source(2)                ' outputs: 3 1 4
Debug.Print destination(0); destination(1); destination(2) ' outputs: 2 1 4
```

Копирование массивов объектов

С массивами объектов копируются *ссылки* на эти объекты, а не сами объекты. Если изменение в объекте в одном массиве будет также изменено в другом массиве - они оба ссылаются на один и тот же объект. Однако установка элемента в другой объект в одном массиве не приведет его к этому объекту в другом массиве.

```
Dim source(0 To 2) As Range
Dim destination() As Range

Set source(0) = Range("A1"): source(0).Value = 3
Set source(1) = Range("A2"): source(1).Value = 1
Set source(2) = Range("A3"): source(2).Value = 4

destination = source

Set destination(0) = Range("A4")   'reference changed in destination but not source

destination(0).Value = 2           'affects an object only in destination
destination(1).Value = 5           'affects an object in both source and destination

Debug.Print source(0); source(1); source(2)                  ' outputs 3 5 4
Debug.Print destination(0); destination(1); destination(2)   ' outputs 2 5 4
```

Варианты, содержащие массив

Вы также можете скопировать массив в переменную варианта и из нее. При копировании из варианта он должен содержать массив того же типа, что и принимающий массив, иначе он будет вызывать ошибку «Ошибка несоответствия типа».

```
Dim var As Variant
Dim source(0 To 2) As Range
Dim destination() As Range

var = source
destination = var

var = 5
destination = var  ' throws runtime error
```

Возвращаемые массивы из функций

Функция в нормальном модуле (но не модуле класса) может возвращать массив, помещая `()` после типа данных.

```
Function arrayOfPiDigits() As Long()
    Dim outputArray(0 To 2) As Long

    outputArray(0) = 3
    outputArray(1) = 1
    outputArray(2) = 4

    arrayOfPiDigits = outputArray
End Function
```

Результат функции можно затем поместить в динамический массив того же типа или варианта. К элементам также можно получить доступ, используя второй набор скобок, однако это вызовет функцию каждый раз, поэтому лучше сохранить результаты в новом массиве, если вы планируете использовать их более одного раза

```
Sub arrayExample()

    Dim destination() As Long
    Dim var As Variant

    destination = arrayOfPiDigits()
    var = arrayOfPiDigits

    Debug.Print destination(0)          ' outputs 3
    Debug.Print var(1)                  ' outputs 1
    Debug.Print arrayOfPiDigits()(2)    ' outputs 4

End Sub
```

Обратите внимание, что возвращаемое на самом деле является копией массива внутри функции, а не ссылкой. Поэтому, если функция возвращает содержимое массива Static, его данные не могут быть изменены процедурой вызова.

Вывод массива через выходной аргумент

Обычно это хорошая практика кодирования для аргументов процедуры как входных данных и для вывода через возвращаемое значение. Однако ограничения VBA иногда требуют, чтобы процедура выводила данные через аргумент `ByRef` .

Вывод в фиксированный массив

```
Sub threePiDigits(ByRef destination() As Long)
    destination(0) = 3
    destination(1) = 1
    destination(2) = 4
End Sub

Sub printPiDigits()
    Dim digits(0 To 2) As Long

    threePiDigits digits
```

```
    Debug.Print digits(0); digits(1); digits(2) ' outputs 3 1 4
End Sub
```

Вывод массива из метода класса

Выходной аргумент также может использоваться для вывода массива из метода / процедуры в модуле класса

```
' Class Module 'MathConstants'
Sub threePiDigits(ByRef destination() As Long)
    ReDim destination(0 To 2)

    destination(0) = 3
    destination(1) = 1
    destination(2) = 4
End Sub

' Standard Code Module
Sub printPiDigits()
    Dim digits() As Long
    Dim mathConsts As New MathConstants

    mathConsts.threePiDigits digits
    Debug.Print digits(0); digits(1); digits(2) ' outputs 3 1 4
End Sub
```

Передача массивов на прохождение

Массивы могут передаваться в процедуры, помещая `()` после имени переменной массива.

```
Function countElements(ByRef arr() As Double) As Long
    countElements = UBound(arr) - LBound(arr) + 1
End Function
```

Массивы *должны* передаваться по ссылке. Если не указан какой-либо передающий механизм, например `myFunction(arr())` , то VBA будет считать `ByRef` по умолчанию, однако хорошая практика кодирования делает его явным. Попытка передать массив по значению, например `myFunction(ByVal arr())` приведет к ошибке компиляции «Array argument должно быть ByRef» (или ошибке компиляции «Синтаксическая ошибка», если `Auto Syntax Check` не проверена в параметрах VBE) ,

Передача по ссылке означает, что любые изменения в массиве будут сохранены в процессе вызова.

```
Sub testArrayPassing()
    Dim source(0 To 1) As Long
    source(0) = 3
    source(1) = 1

    Debug.Print doubleAndSum(source)  ' outputs 8
    Debug.Print source(0); source(1)  ' outputs 6 2
```

```
End Sub

Function doubleAndSum(ByRef arr() As Long)
    arr(0) = arr(0) * 2
    arr(1) = arr(1) * 2
    doubleAndSum = arr(0) + arr(1)
End Function
```

Если вы хотите избежать изменения исходного массива, будьте осторожны, чтобы написать функцию, чтобы она не меняла никаких элементов.

```
Function doubleAndSum(ByRef arr() As Long)
    doubleAndSum = arr(0) * 2 + arr(1) * 2
End Function
```

Альтернативно создайте рабочую копию массива и работайте с копией.

```
Function doubleAndSum(ByRef arr() As Long)
    Dim copyOfArr() As Long
    copyOfArr = arr

    copyOfArr(0) = copyOfArr(0) * 2
    copyOfArr(1) = copyOfArr(1) * 2

    doubleAndSum = copyOfArr(0) + copyOfArr(1)
End Function
```

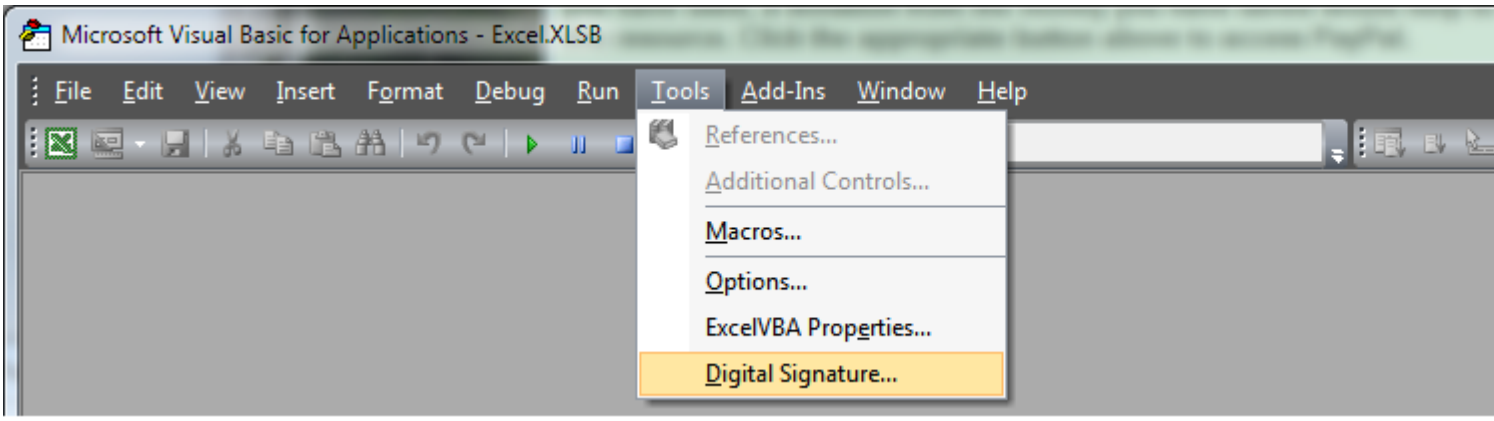
Прочитайте Копирование, возврат и передача массивов онлайн:
https://riptutorial.com/ru/vba/topic/9069/копирование--возврат-и-передача-массивов

глава 14: Макрозащита и подписание VBA-проектов / -модулей

Examples

Создайте действительный цифровой самозаверяющий сертификат SELFCERT.EXE

Для запуска макросов и обеспечения безопасности приложений Office, предоставляемых против вредоносного кода, необходимо цифровое подписание VBAProject.OTM из *редактора VBA> Инструменты> Цифровая подпись* .

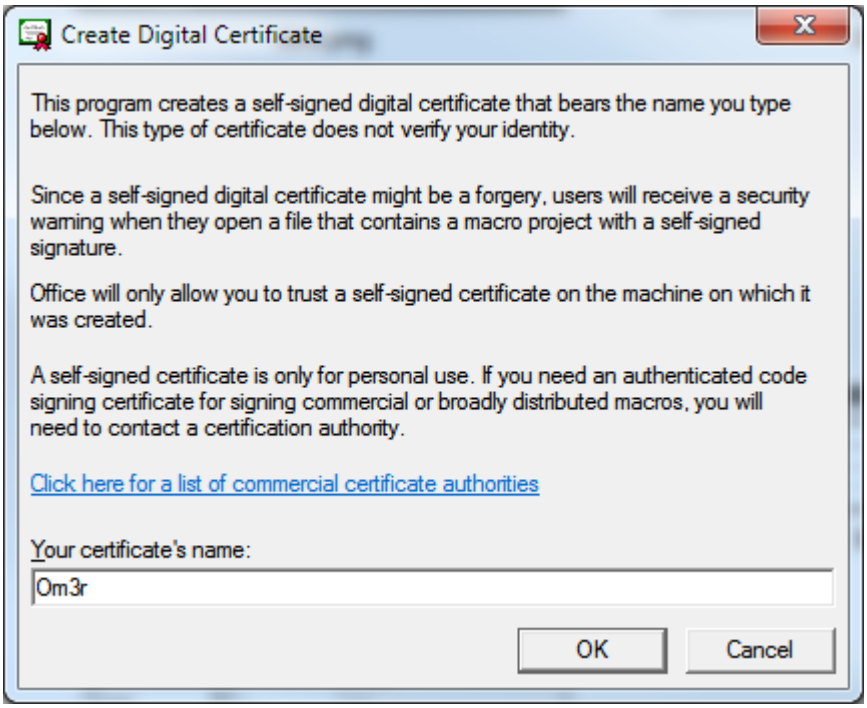


Office поставляется с утилитой для создания самозаверяющего цифрового сертификата, который вы можете использовать на ПК для подписания ваших проектов.

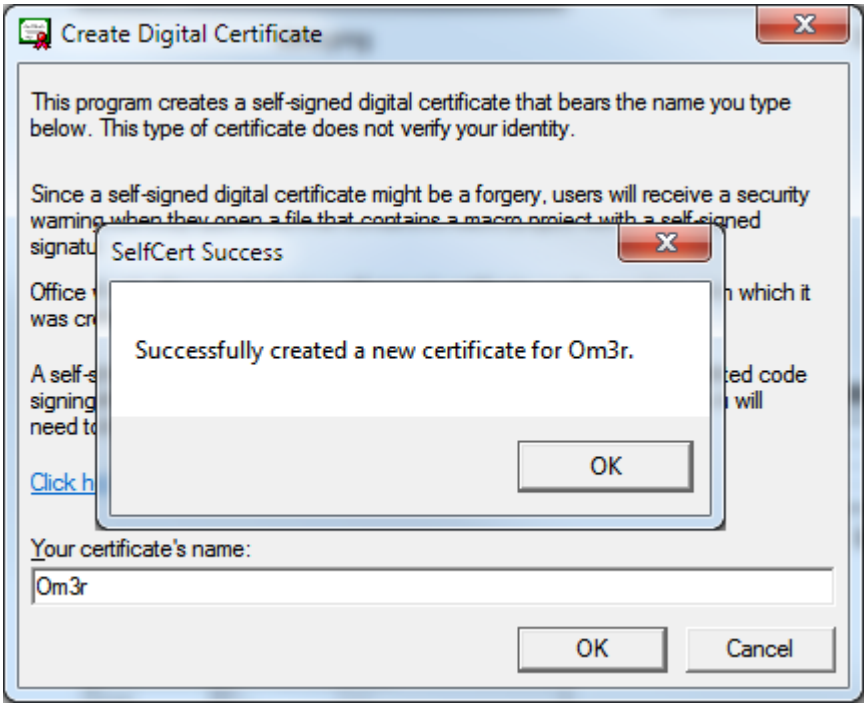
Эта утилита **SELFCERT.EXE** находится в папке программы Office,

Нажмите «Цифровой сертификат для проектов VBA», чтобы открыть *мастер* сертификатов.

В диалоговом окне введите подходящее имя для сертификата и нажмите «OK».

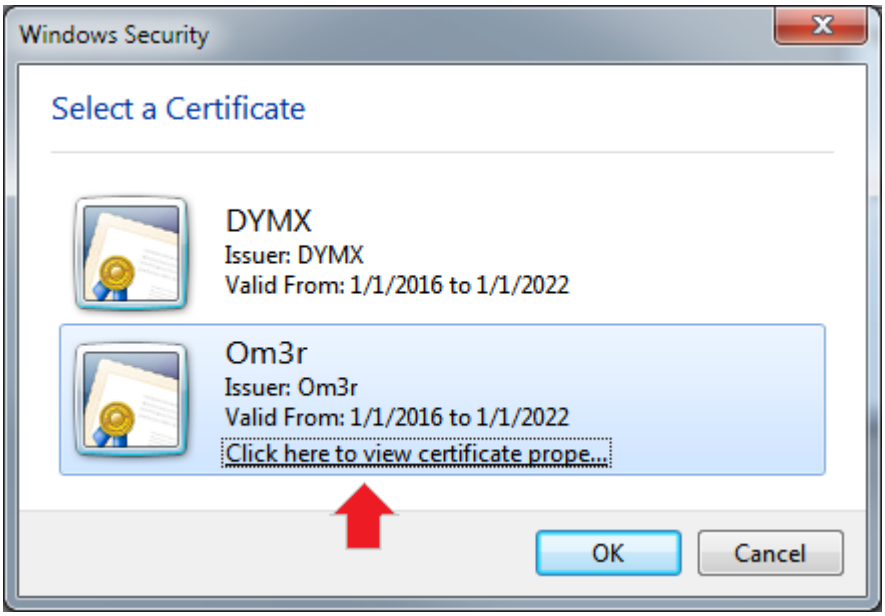


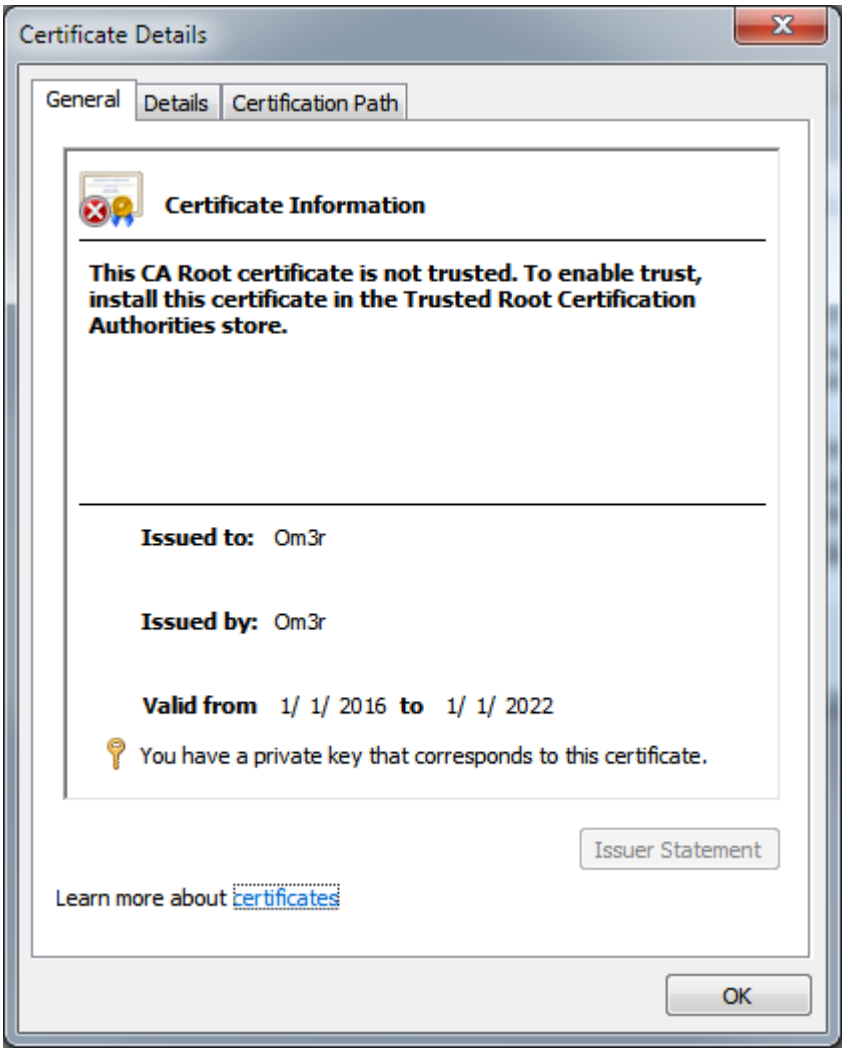
Если все пойдет хорошо, вы увидите подтверждение:



Теперь вы можете закрыть мастер **SELFCERT** и обратить ваше внимание на созданный сертификат.

Если вы попытаетесь использовать сертификат, который вы только что создали, и вы проверяете его свойства

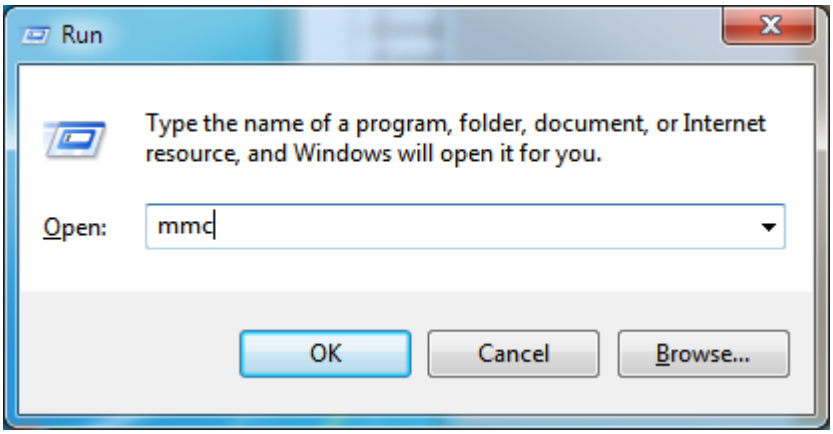




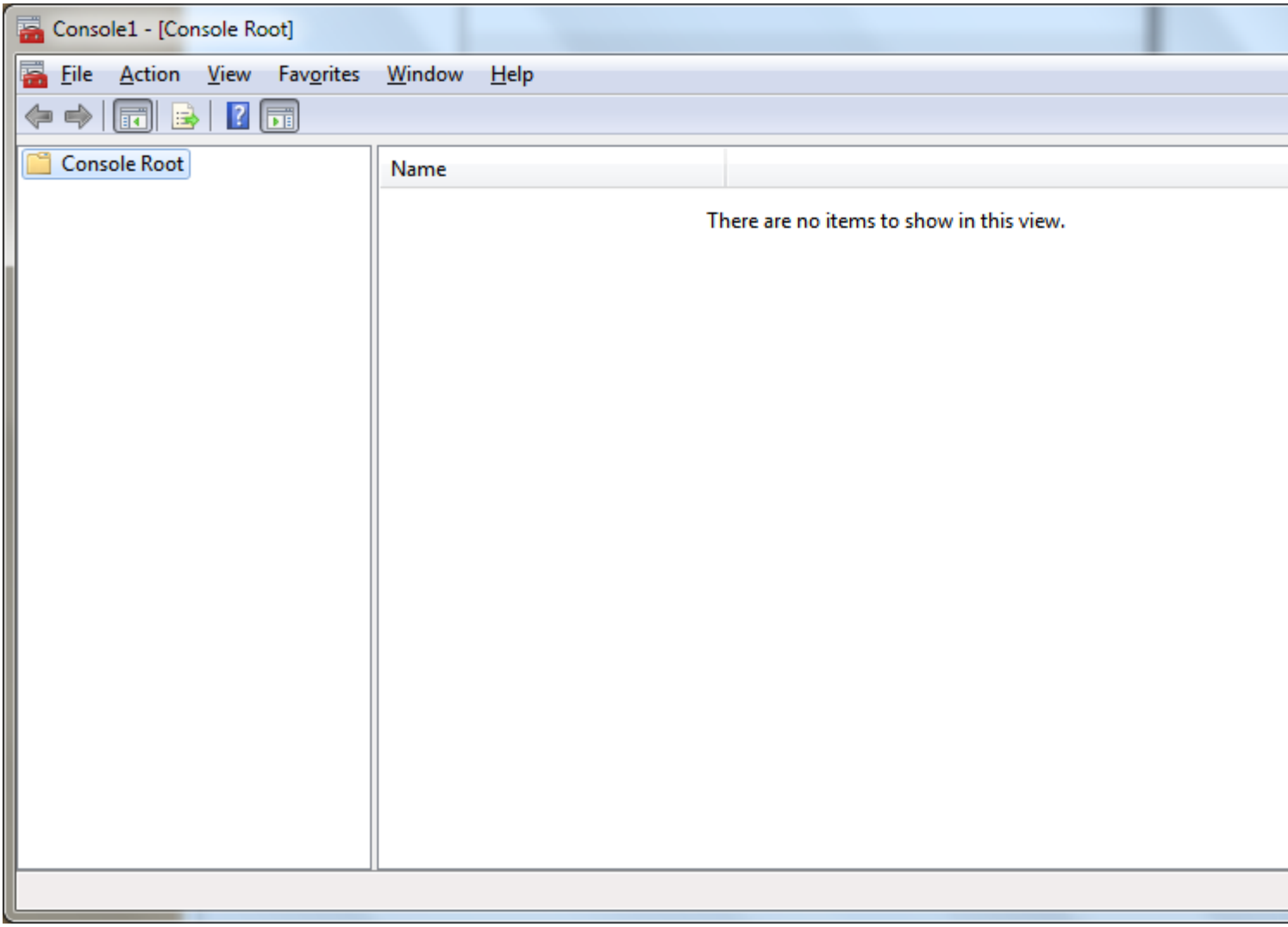
Вы увидите, что сертификат не доверен, и причина указана в диалоговом окне.

Сертификат создан в хранилище «Текущий пользователь»> «Личное»> «Сертификаты». Он должен находиться в локальном компьютере> Доверенные корневые центры сертификации> Хранилища сертификатов, поэтому вам нужно экспортировать из первого и импортировать его в последний.

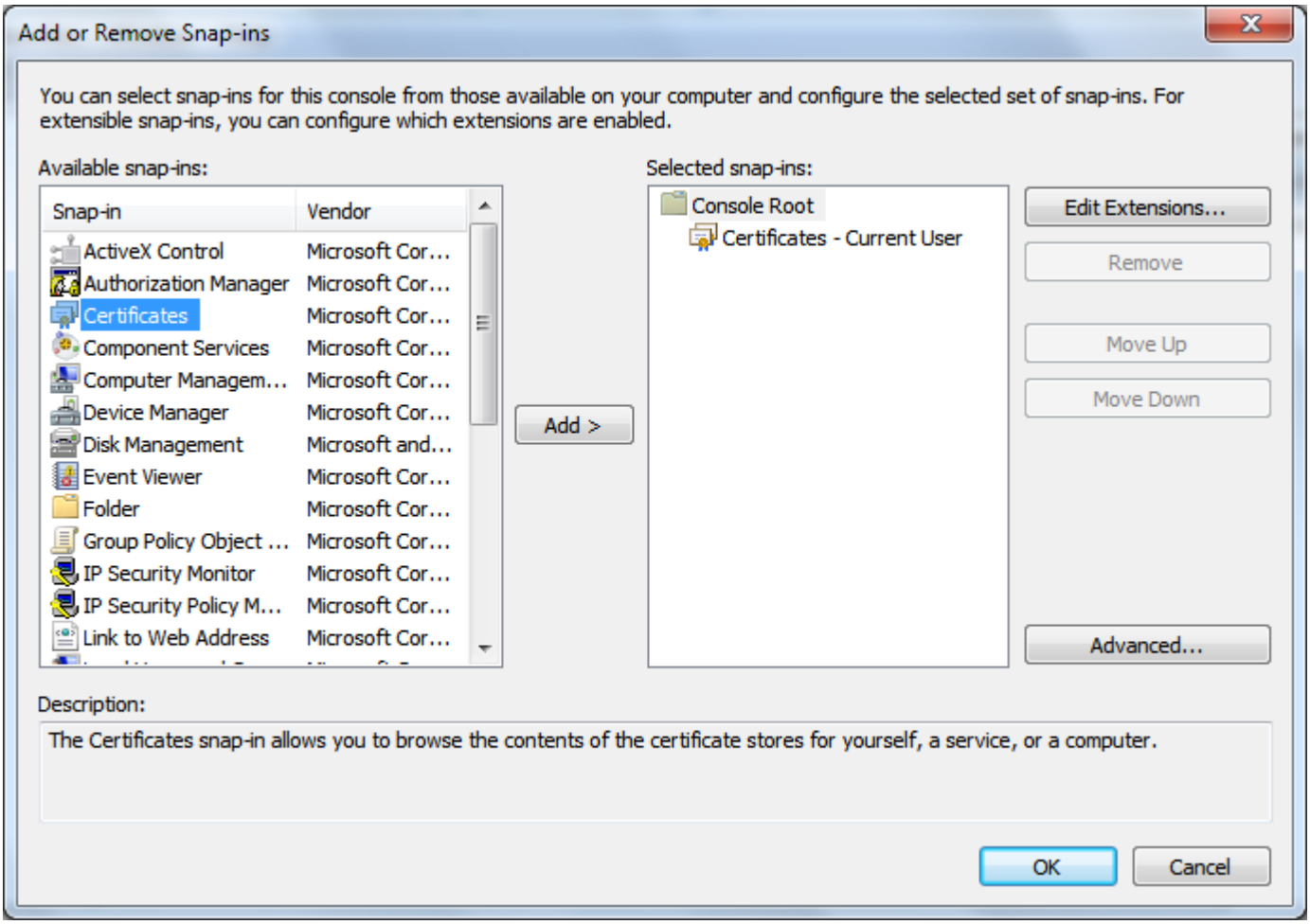
Нажатие клавиши Windows + R, которая откроет окно «Выполнить». затем введите «mmc» в окне, как показано ниже, и нажмите «OK».



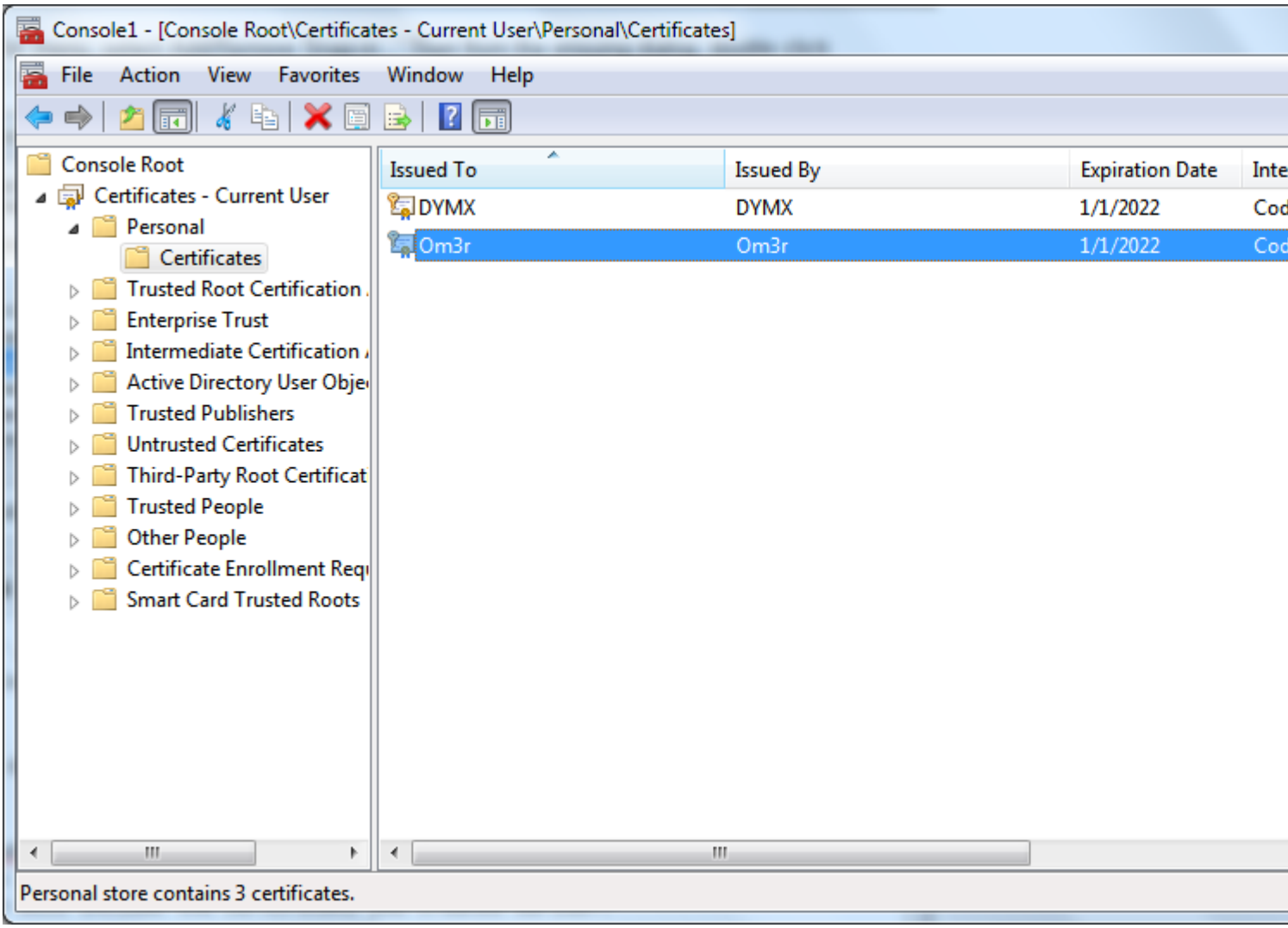
Консоль управления Microsoft откроется и будет выглядеть следующим образом.



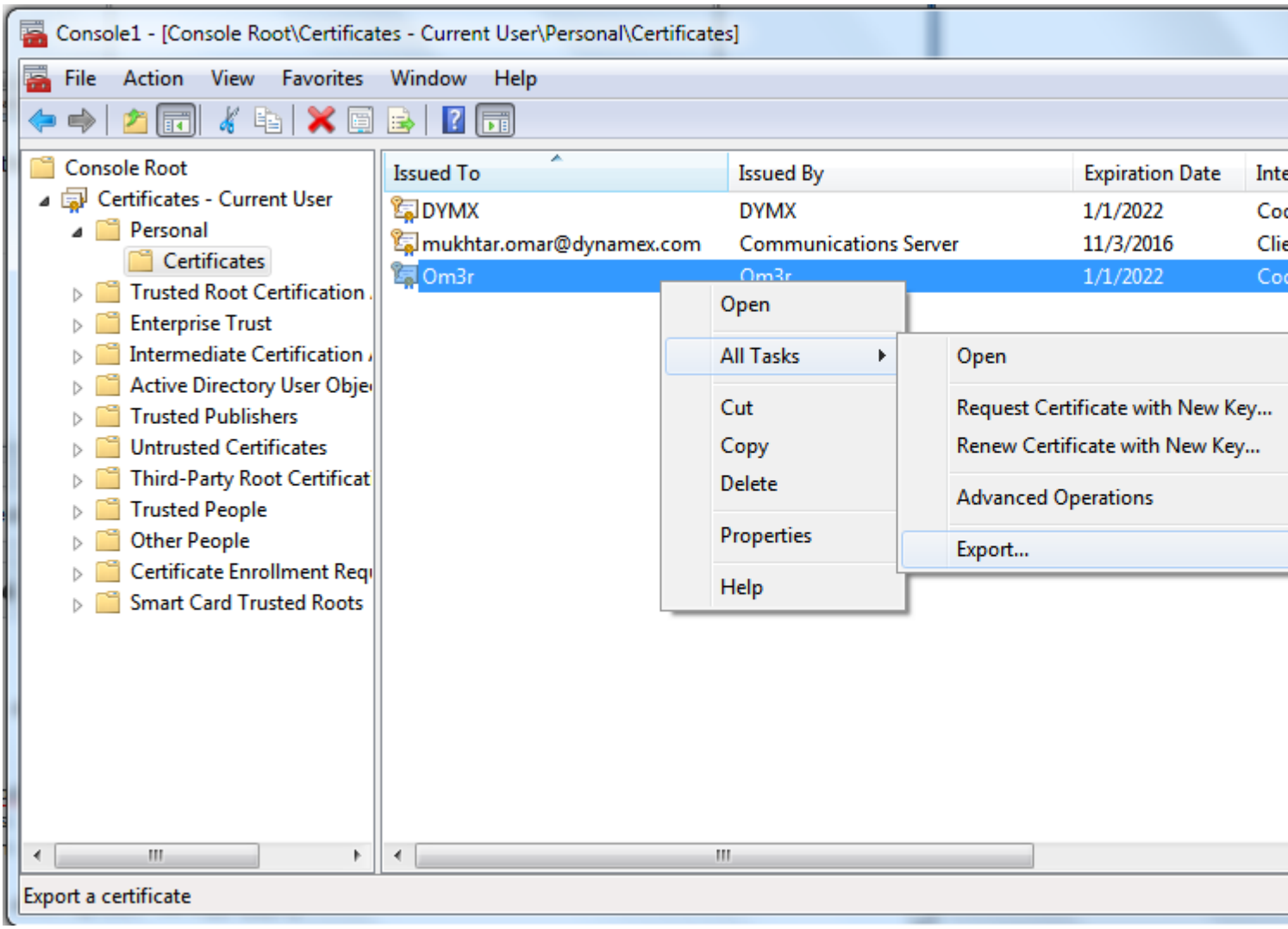
В меню «Файл» выберите «Добавить / удалить оснастку» ... Затем в появившемся диалоговом окне дважды щелкните «Сертификаты» и нажмите «OK».



Разверните раскрывающийся список в левом окне для « *Сертификаты - текущий пользователь* » и выберите сертификаты, как показано ниже. Затем центральная панель отобразит сертификаты в этом месте, которые будут содержать сертификат, который вы создали ранее:

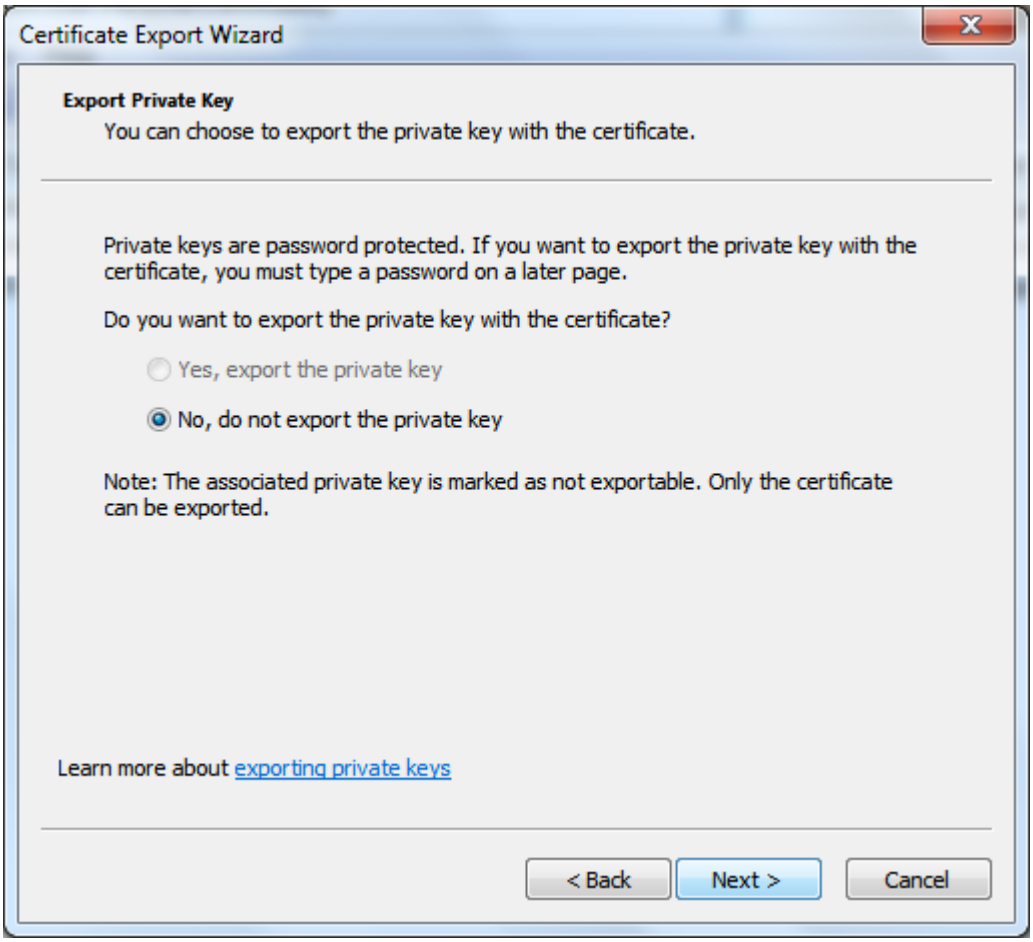


Щелкните правой кнопкой мыши сертификат и выберите «Все задачи»> «Экспорт»:

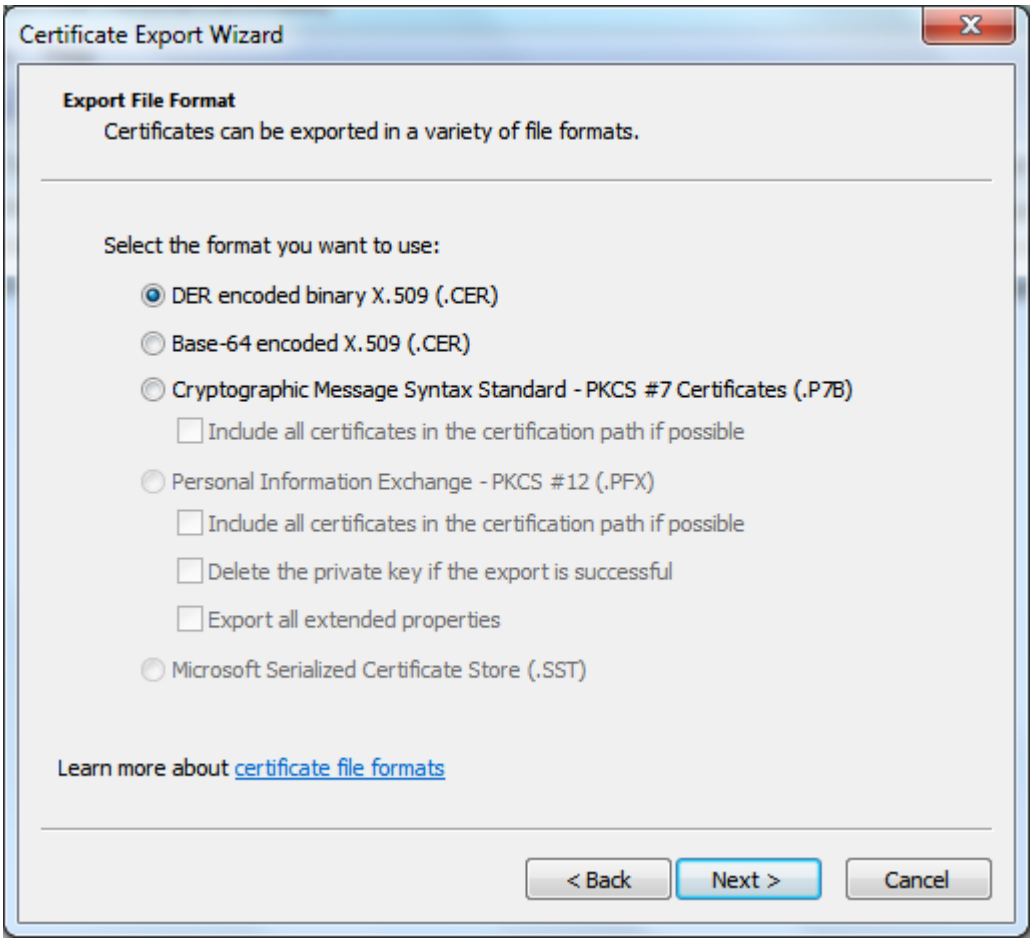


Мастер экспорта

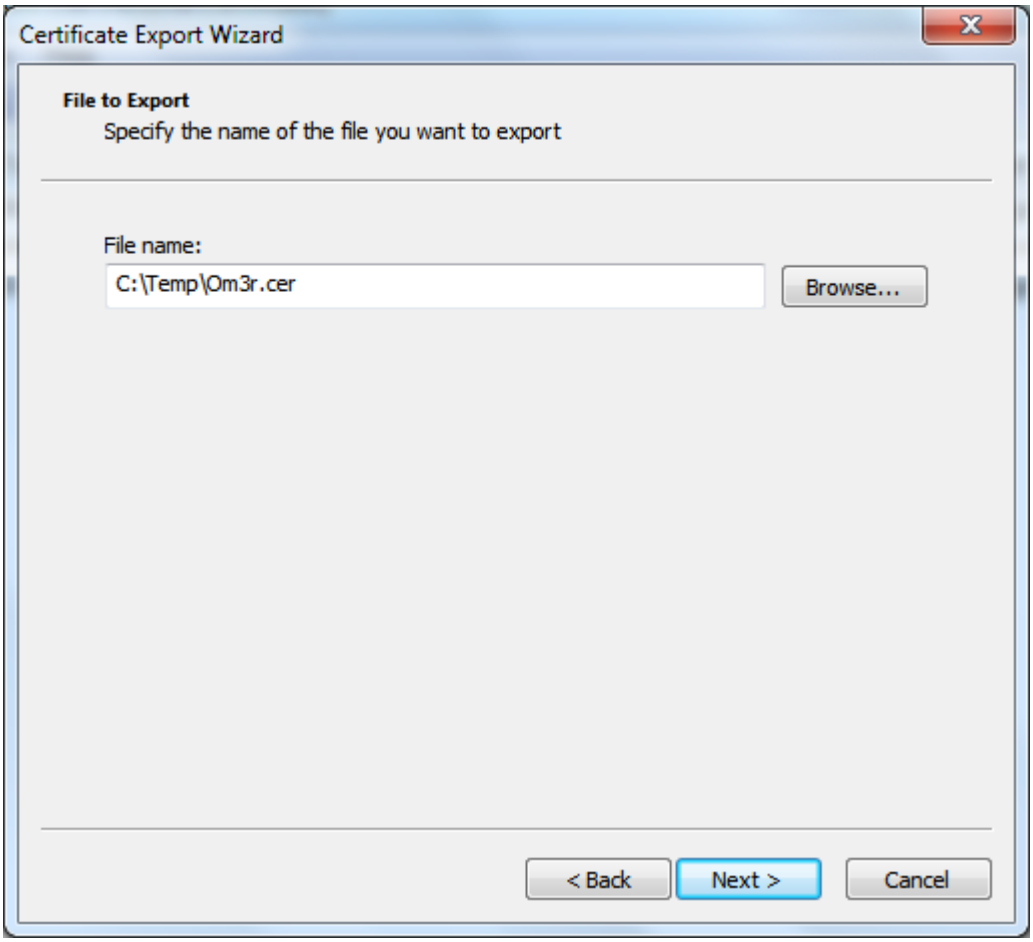
Нажмите кнопку "Далее



Будет доступен только один предварительно выбранный вариант, поэтому нажмите «Далее» еще раз:



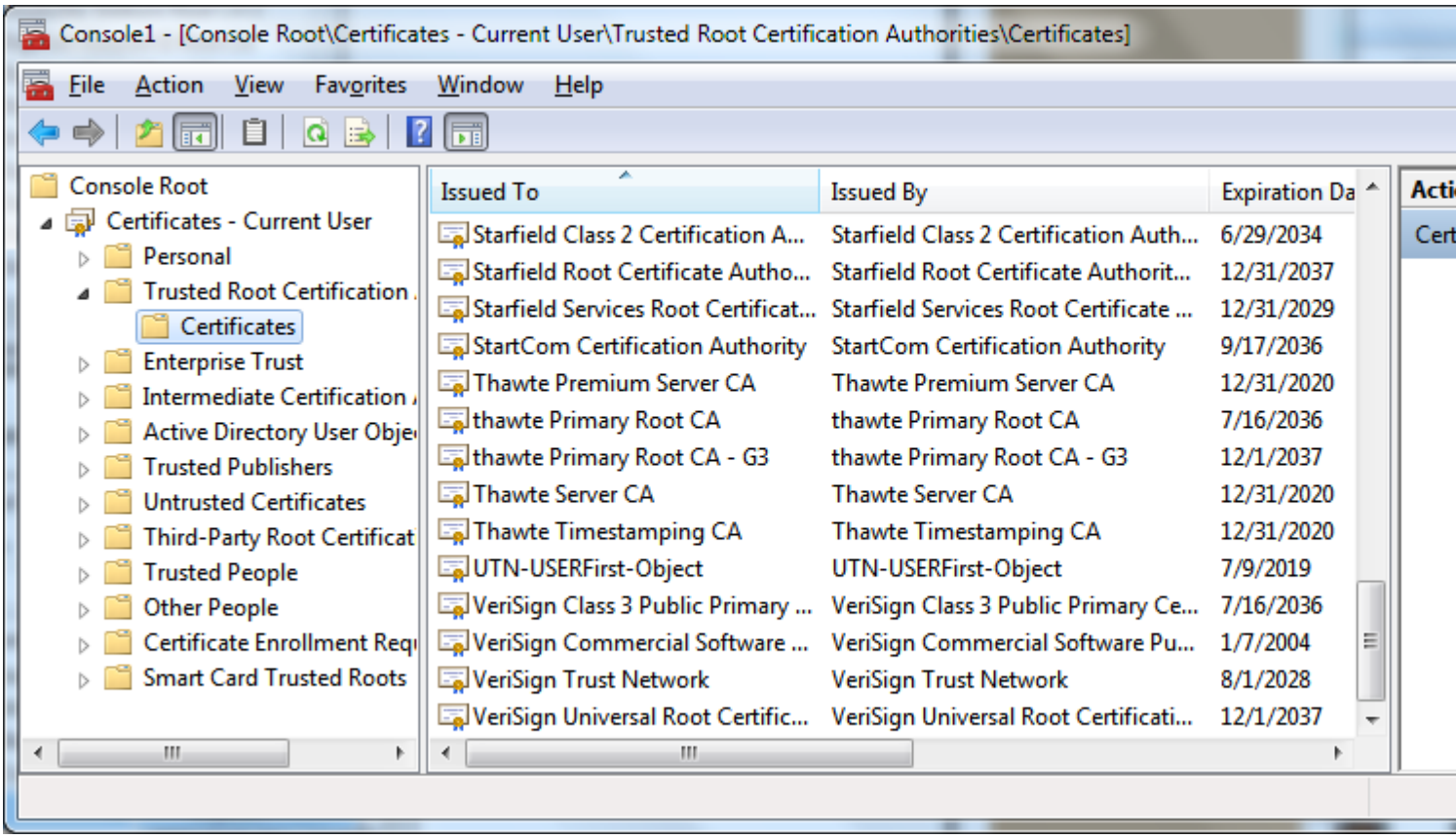
Верхний элемент уже будет предварительно выбран. Нажмите «Далее» еще раз и выберите имя и местоположение, чтобы сохранить экспортированный сертификат.



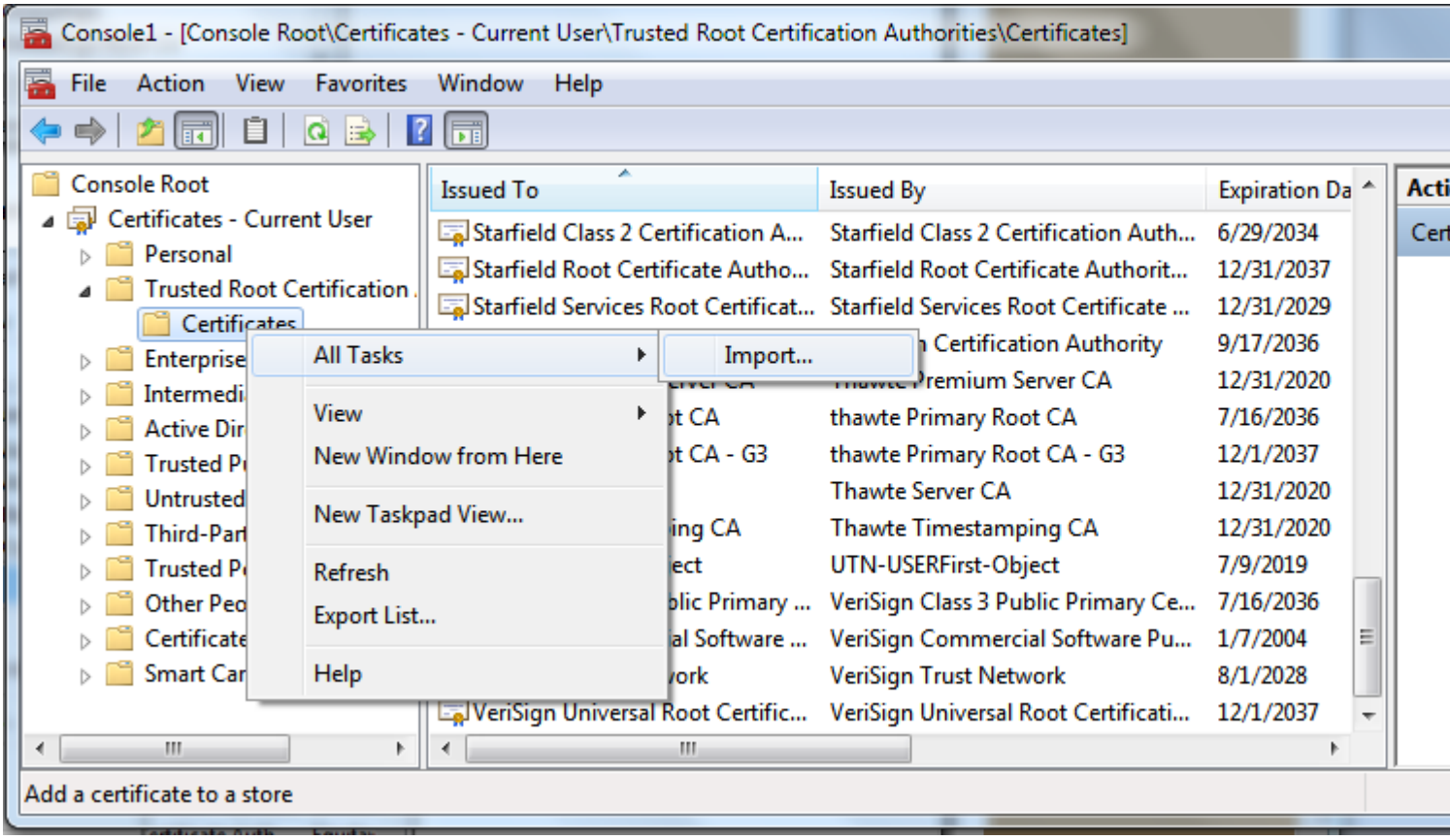
Нажмите «Далее» еще раз, чтобы сохранить сертификат.

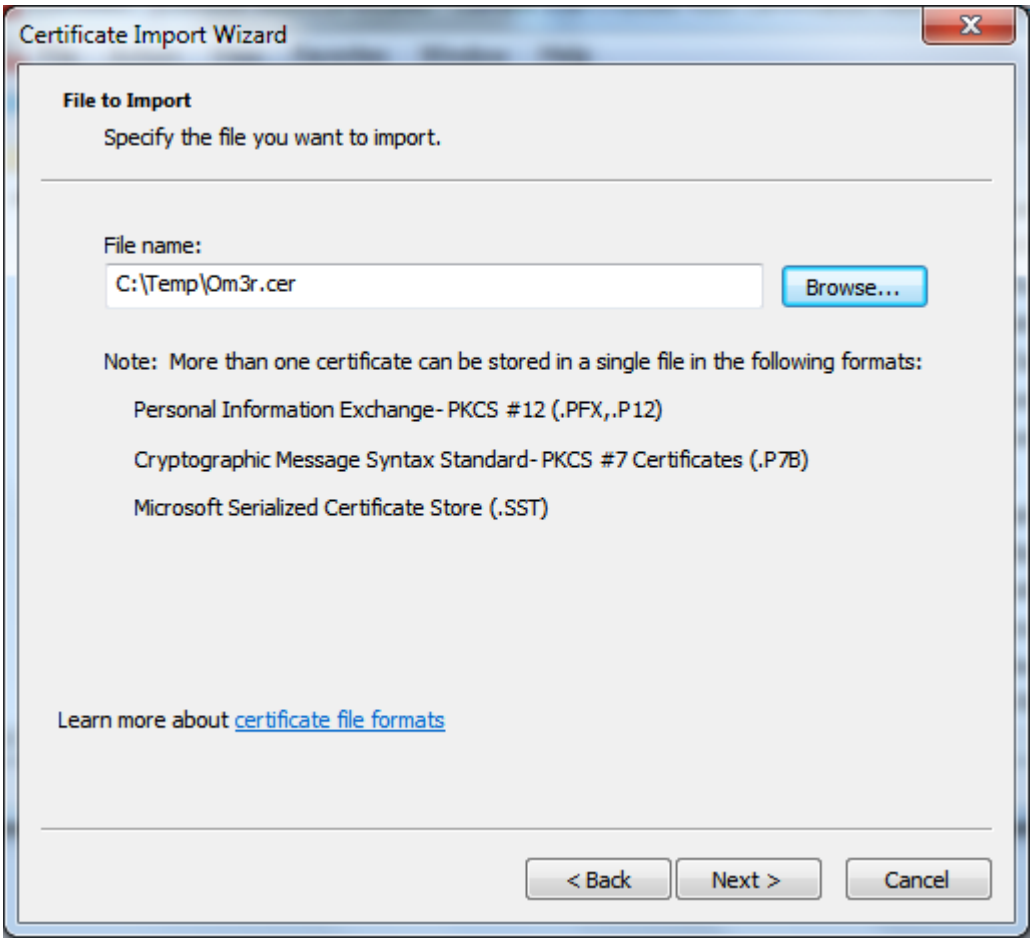
Как только фокус возвращается в Консоль управления.

Разверните меню « *Сертификаты» и в* меню «Надежные корневые центры сертификации» выберите « *Сертификаты»* .

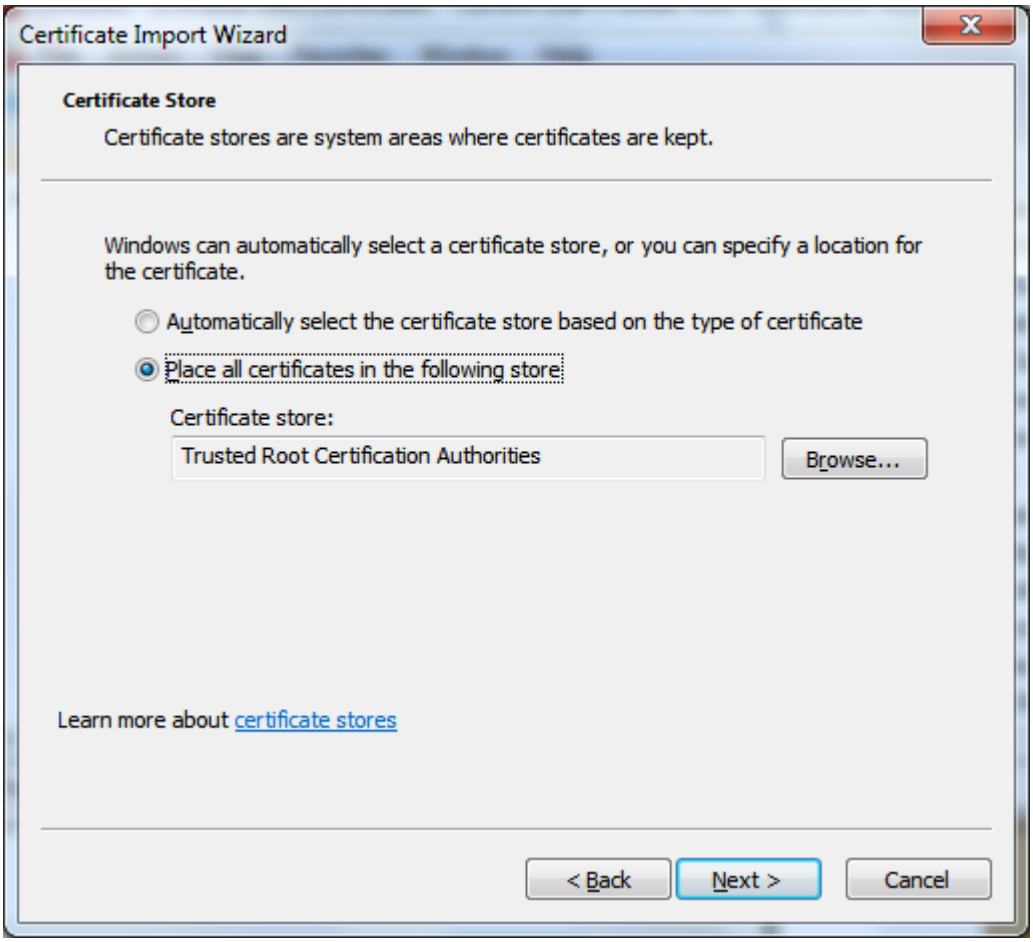


Щелкните правой кнопкой мыши. Выбрать *все задачи* и *импорт*



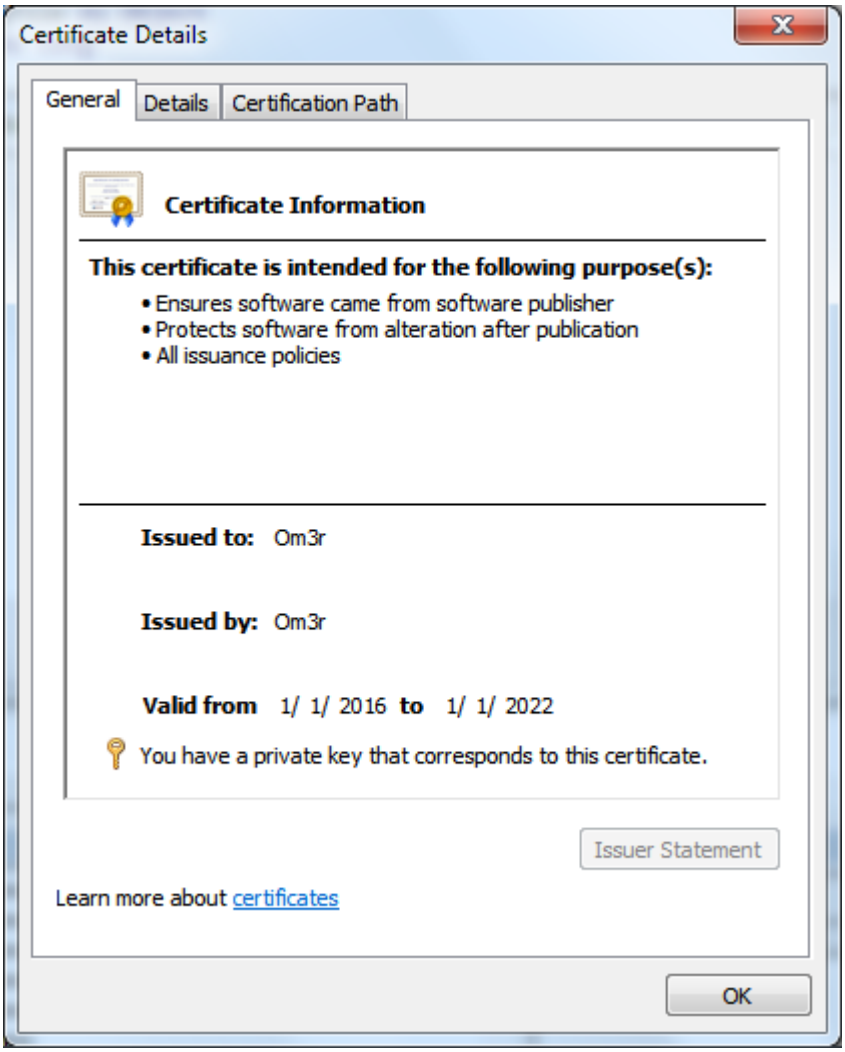


Нажмите «Далее» и «Сохранить» в *хранилище доверенных корневых центров сертификации* :



Затем Next> Finish, теперь закройте консоль.

Если вы теперь используете сертификат и проверяете его свойства, вы увидите, что он является доверенным сертификатом, и вы можете использовать его для подписания вашего проекта:



Прочитайте Макрозащита и подписание VBA-проектов / -модулей онлайн:
https://riptutorial.com/ru/vba/topic/7733/макрозащита-и-подписание-vba-проектов----модулей

глава 15: Манипуляция времени по времени

Examples

Календарь

VBA поддерживает 2 календаря: григорианский и хиджры

Свойство `Calendar` используется для изменения или отображения текущего календаря.

2 значения для Календаря:

Значение	постоянная	Описание
0	vbCalGreg	Григорианский календарь (по умолчанию)
1	vbCalHijri	Календарь Hijri

пример

```
Sub CalendarExample()
    'Cache the current setting.
    Dim Cached As Integer
    Cached = Calendar

    ' Dates in Gregorian Calendar
    Calendar = vbCalGreg
    Dim Sample As Date
    'Create sample date of 2016-07-28
    Sample = DateSerial(2016, 7, 28)

    Debug.Print "Current Calendar : " & Calendar
    Debug.Print "SampleDate = " & Format$(Sample, "yyyy-mm-dd")

    ' Date in Hijri Calendar
    Calendar = vbCalHijri
    Debug.Print "Current Calendar : " & Calendar
    Debug.Print "SampleDate = " & Format$(Sample, "yyyy-mm-dd")

    'Reset VBA to cached value.
    Cached = Calendar
End Sub
```

Этот Sub печатает следующее:

```
Current Calendar : 0
```

```
SampleDate = 2016-07-28
Current Calendar : 1
SampleDate = 1437-10-23
```

Базовые функции

Получить систему DateTime

VBA поддерживает 3 встроенные функции для извлечения даты и / или времени из часов системы.

функция	Тип возврата	Возвращаемое значение
Сейчас	Дата	Возвращает текущую дату и время
Дата	Дата	Возвращает часть даты текущей даты и времени
Время	Дата	Возвращает временную часть текущей даты и времени

```
Sub DateTimeExample()

    ' -----------------------------------------------------
    ' Note : EU system with default date format DD/MM/YYYY
    ' -----------------------------------------------------

    Debug.Print Now   ' prints 28/07/2016 10:16:01 (output below assumes this date and time)
    Debug.Print Date  ' prints 28/07/2016
    Debug.Print Time  ' prints 10:16:01

    ' Apply a custom format to the current date or time
    Debug.Print Format$(Now, "dd mmmm yyyy hh:nn")  ' prints 28 July 2016 10:16
    Debug.Print Format$(Date, "yyyy-mm-dd")         ' prints 2016-07-28
    Debug.Print Format$(Time, "hh") & " hour " & _
                Format$(Time, "nn") & " min " & _
                Format$(Time, "ss") & " sec "       ' prints 10 hour 16 min 01 sec

End Sub
```

Функция таймера

Функция `Timer` возвращает значение Single, представляющее количество секунд, прошедших с полуночи. Точность составляет сотую доли секунды.

```
Sub TimerExample()

    Debug.Print Time    ' prints 10:36:31  (time at execution)
    Debug.Print Timer   ' prints 38191,13  (seconds since midnight)

End Sub
```

Поскольку функции «В `Now` `Time` и « `Time` являются точными до нескольких секунд, `Timer` предлагает удобный способ повысить точность измерения времени:

```
Sub GetBenchmark()

    Dim StartTime  As Single
    StartTime = Timer       'Store the current Time

    Dim i As Long
    Dim temp As String
    For i = 1 To 1000000    'See how long it takes Left$ to execute 1,000,000 times
        temp = Left$("Text", 2)
    Next i

    Dim Elapsed As Single
    Elapsed = Timer - StartTime
    Debug.Print "Code completed in " & CInt(Elapsed * 1000) & " ms"

End Sub
```

IsDate ()

IsDate () проверяет, является ли выражение допустимой датой или нет. Возвращает `Boolean` .

```
Sub IsDateExamples()

    Dim anything As Variant

    anything = "September 11, 2001"

    Debug.Print IsDate(anything)    'Prints True

    anything = #9/11/2001#

    Debug.Print IsDate(anything)    'Prints True

    anything = "just a string"

    Debug.Print IsDate(anything)    'Prints False

    anything = vbNull

    Debug.Print IsDate(anything)    'Prints False

End Sub
```

Функции экстракции

Эти функции принимают `Variant` который может быть передан в `Date` как параметр и возвращает `Integer` представляющее часть даты или времени. Если параметр не может быть применен к `Date` , это приведет к ошибке времени выполнения 13: несоответствие типа.

функция	Описание	Возвращаемое значение
Год()	Возвращает часть года аргумента даты.	Целое число (от 100 до 9999)
Месяц()	Возвращает месячную часть аргумента date.	Целое число (от 1 до 12)
День()	Возвращает дневную часть аргумента date.	Целое число (от 1 до 31)
WeekDay ()	Возвращает день недели аргумента даты. Принимает необязательный второй аргумент, определяющий первый день недели	Целое число (от 1 до 7)
Час()	Возвращает часовую часть аргумента date.	Целое число (от 0 до 23)
Минута ()	Возвращает минутную часть аргумента date.	Целое число (от 0 до 59)
Во-вторых ()	Возвращает вторую часть аргумента date.	Целое число (от 0 до 59)

Примеры:

```
Sub ExtractionExamples()

    Dim MyDate As Date

    MyDate = DateSerial(2016, 7, 28) + TimeSerial(12, 34, 56)

    Debug.Print Format$(MyDate, "yyyy-mm-dd hh:nn:ss") ' prints 2016-07-28 12:34:56

    Debug.Print Year(MyDate)                           ' prints 2016
    Debug.Print Month(MyDate)                          ' prints 7
    Debug.Print Day(MyDate)                            ' prints 28
    Debug.Print Hour(MyDate)                           ' prints 12
    Debug.Print Minute(MyDate)                         ' prints 34
    Debug.Print Second(MyDate)                         ' prints 56

    Debug.Print Weekday(MyDate)                        ' prints 5
    'Varies by locale - i.e. will print 4 in the EU and 5 in the US
    Debug.Print Weekday(MyDate, vbUseSystemDayOfWeek)
    Debug.Print Weekday(MyDate, vbMonday)              ' prints 4
    Debug.Print Weekday(MyDate, vbSunday)              ' prints 5

End Sub
```

Функция DatePart ()

`DatePart()` также является функцией, возвращающей часть даты, но работает по-разному и предоставляет больше возможностей, чем функции выше. Он может, например, вернуться в квартал года или в Неделю года.

Синтаксис:

```
DatePart ( interval, date  [, firstdayofweek] [, firstweekofyear] )
```

интервальный аргумент может быть:

интервал	Описание
«Гггг»	Год (от 100 до 9999)
«У»	День года (от 1 до 366)
«М»	Месяц (от 1 до 12)
«Д»	Четверть (от 1 до 4)
"WW"	Неделя (от 1 до 53)
«Ж»	День недели (от 1 до 7)
«Г»	День месяца (от 1 до 31)
"час"	Час (от 0 до 23)
«П»	Минута (от 0 до 59)
"S"	Второй (от 0 до 59)

firstdayofweek не является обязательным. это константа, указывающая первый день недели. Если не указано, предполагается `vbSunday` .

firstweekofyear не является обязательным. это константа, указывающая первую неделю года. Если не указано, первая неделя считается неделей, в которой происходит 1 января.

Примеры:

```
Sub DatePartExample()

    Dim MyDate As Date

    MyDate = DateSerial(2016, 7, 28) + TimeSerial(12, 34, 56)
```

```
    Debug.Print Format$(MyDate, "yyyy-mm-dd hh:nn:ss") ' prints 2016-07-28 12:34:56

    Debug.Print DatePart("yyyy", MyDate)              ' prints 2016
    Debug.Print DatePart("y", MyDate)                 ' prints 210
    Debug.Print DatePart("h", MyDate)                 ' prints 12
    Debug.Print DatePart("Q", MyDate)                 ' prints 3
    Debug.Print DatePart("w", MyDate)                 ' prints 5
    Debug.Print DatePart("ww", MyDate)                ' prints 31

End Sub
```

Функции вычисления

DateDiff ()

`DateDiff()` возвращает `Long` представляющий количество интервалов времени между двумя указанными датами.

Синтаксис

```
DateDiff ( interval, date1, date2  [, firstdayofweek] [, firstweekofyear] )
```

- *интервал* может быть любым из интервалов, определенных в функции `DatePart()`
- *date1* и *date2* являются две даты , которые вы хотите использовать в расчетах
- *firstdayofweek* и *firstweekofyear* являются необязательными. Обратитесь к функции `DatePart()` для объяснений

Примеры

```
Sub DateDiffExamples()

    ' Check to see if 2016 is a leap year.
    Dim NumberOfDays As Long
    NumberOfDays = DateDiff("d", #1/1/2016#, #1/1/2017#)

    If NumberOfDays = 366 Then
        Debug.Print "2016 is a leap year."              'This will output.
    End If

    ' Number of seconds in a day
    Dim StartTime As Date
    Dim EndTime As Date
    StartTime = TimeSerial(0, 0, 0)
    EndTime = TimeSerial(24, 0, 0)
    Debug.Print DateDiff("s", StartTime, EndTime)       'prints 86400

End Sub
```

DateAdd ()

`DateAdd()` возвращает `Date` к которой была добавлена указанная дата или временной интервал.

Синтаксис

```
DateAdd ( interval, number, date  )
```

- *интервал* может быть любым из интервалов, определенных в функции `DatePart()`
- *number* Числовое выражение, которое представляет собой количество интервалов, которые вы хотите добавить. Это может быть положительным (для получения дат в будущем) или отрицательным (для получения дат в прошлом).
- *date* - `Date` или литерал, представляющий дату, в которую добавлен интервал

Примеры :

```
Sub DateAddExamples()

    Dim Sample As Date
    'Create sample date and time of 2016-07-28 12:34:56
    Sample = DateSerial(2016, 7, 28) + TimeSerial(12, 34, 56)

    ' Date 5 months previously (prints 2016-02-28):
    Debug.Print Format$(DateAdd("m", -5, Sample), "yyyy-mm-dd")

    ' Date 10 months previously (prints 2015-09-28):
    Debug.Print Format$(DateAdd("m", -10, Sample), "yyyy-mm-dd")

    ' Date in 8 months (prints 2017-03-28):
    Debug.Print Format$(DateAdd("m", 8, Sample), "yyyy-mm-dd")

    ' Date/Time 18 hours previously (prints 2016-07-27 18:34:56):
    Debug.Print Format$(DateAdd("h", -18, Sample), "yyyy-mm-dd hh:nn:ss")

    ' Date/Time in 36 hours (prints 2016-07-30 00:34:56):
    Debug.Print Format$(DateAdd("h", 36, Sample), "yyyy-mm-dd hh:nn:ss")

End Sub
```

Преобразование и создание

CDate ()

`CDate()` преобразует что-то из любого типа данных в тип данных `Date`

```
Sub CDateExamples()

    Dim sample As Date

    ' Converts a String representing a date and time to a Date
    sample = CDate("September 11, 2001 12:34")
    Debug.Print Format$(sample, "yyyy-mm-dd hh:nn:ss")      ' prints 2001-09-11 12:34:00
```

```
    ' Converts a String containing a date to a Date
    sample = CDate("September 11, 2001")
    Debug.Print Format$(sample, "yyyy-mm-dd hh:nn:ss")      ' prints 2001-09-11 00:00:00

    ' Converts a String containing a time to a Date
    sample = CDate("12:34:56")
    Debug.Print Hour(sample)                        ' prints 12
    Debug.Print Minute(sample)                      ' prints 34
    Debug.Print Second(sample)                      ' prints 56

    ' Find the 10000th day from the epoch date of 1899-12-31
    sample = CDate(10000)
    Debug.Print Format$(sample, "yyyy-mm-dd")       ' prints 1927-05-18

End Sub
```

Обратите внимание, что VBA также имеет слабо типизированный `CVDate()` который функционирует так же, как и `CDate()` от возвращаемого `Variant` типизированной датой вместо строго типизированной `Date` . Версия `CDate()` должна быть предпочтительной при переходе к параметру `Date` или присвоении переменной `Date` , а `CVDate()` должна быть предпочтительной при переходе к параметру `Variant` или присвоении переменной `Variant` . Это позволяет избежать неявного литья типов.

DateSerial ()

`DateSerial()` используется для создания даты. Он возвращает `Date` для указанного года, месяца и дня.

Синтаксис:

```
DateSerial ( year, month, day )
```

Имеются действительные аргументы год, месяц и день. Целые числа (год от 100 до 9999, месяц от 1 до 12, день от 1 до 31).

Примеры

```
Sub DateSerialExamples()

    ' Build a specific date
    Dim sample As Date
    sample = DateSerial(2001, 9, 11)
    Debug.Print Format$(sample, "yyyy-mm-dd")                   ' prints 2001-09-11

    ' Find the first day of the month for a date.
    sample = DateSerial(Year(sample), Month(sample), 1)
    Debug.Print Format$(sample, "yyyy-mm-dd")                   ' prints 2001-09-11

    ' Find the last day of the previous month.
    sample = DateSerial(Year(sample), Month(sample), 1) - 1
    Debug.Print Format$(sample, "yyyy-mm-dd")                   ' prints 2001-09-11
```

```
End Sub
```

Обратите внимание, что `DateSerial()` принимает «недействительные» даты и вычисляет действительную дату из него. Это можно использовать творчески для хорошего:

Положительный пример

```
Sub GoodDateSerialExample()

    'Calculate 45 days from today
    Dim today As Date
    today = DateSerial (2001, 9, 11)
    Dim futureDate As Date
    futureDate = DateSerial(Year(today), Month(today), Day(today) + 45)
    Debug.Print Format$(futureDate, "yyyy-mm-dd")            'prints 2009-10-26

End Sub
```

Однако при попытке создать дату из неутвержденного пользовательского ввода чаще возникает гореча:

Отрицательный пример

```
Sub BadDateSerialExample()

    'Allow user to enter unvalidate date information
    Dim myYear As Long
    myYear = InputBox("Enter Year")
            'Assume user enters 2009
    Dim myMonth As Long
    myMonth = InputBox("Enter Month")
            'Assume user enters 2
    Dim myDay As Long
    myDay = InputBox("Enter Day")
            'Assume user enters 31
    Debug.Print Format$(DateSerial(myYear, myMonth, myDay), "yyyy-mm-dd")
            'prints  2009-03-03

End Sub
```

Прочитайте Манипуляция времени по времени онлайн:
https://riptutorial.com/ru/vba/topic/4452/манипуляция-времени-по-времени

глава 16: Массивы

Examples

Объявление массива в VBA

Объявление массива очень похоже на объявление переменной, за исключением того, что вам нужно объявить размер массива сразу после его имени:

```
Dim myArray(9) As String 'Declaring an array that will contain up to 10 strings
```

По умолчанию массивы в VBA **индексируются из ZERO** , поэтому число внутри скобки не относится к размеру массива, а скорее к **индексу последнего элемента**

Доступ к элементам

Доступ к элементу массива осуществляется с использованием имени массива, за которым следует индекс элемента, внутри скобки:

```
myArray(0) = "first element"
myArray(5) = "sixth element"
myArray(9) = "last element"
```

Индексирование массива

Вы можете изменить индексирование массивов, разместив эту строку в верхней части модуля:

```
Option Base 1
```

С этой строкой все объявленные в модуле массивы будут **проиндексированы с ONE** .

Специфический указатель

Вы также можете объявить каждый массив своим собственным индексом, используя ключевое слово `To` , а нижнюю и верхнюю границы (= индекс):

```
Dim mySecondArray(1 To 12) As String 'Array of 12 strings indexed from 1 to 12
Dim myThirdArray(13 To 24) As String 'Array of 12 strings indexed from 13 to 24
```

Динамическая декларация

Когда вы не знаете размер своего массива до его объявления, вы можете использовать динамическое объявление и ключевое слово `ReDim` :

```
Dim myDynamicArray() As Strings 'Creates an Array of an unknown number of strings
ReDim myDynamicArray(5) 'This resets the array to 6 elements
```

Обратите внимание, что использование ключевого слова `ReDim` уничтожит любое предыдущее содержимое вашего массива. Чтобы предотвратить это, вы можете использовать ключевое слово `Preserve` после `ReDim` :

```
Dim myDynamicArray(5) As String
myDynamicArray(0) = "Something I want to keep"

ReDim Preserve myDynamicArray(8) 'Expand the size to up to 9 strings
Debug.Print myDynamicArray(0) ' still prints the element
```

Использование Split для создания массива из строки

Функция разделения

возвращает одномерный массив на основе нуля, содержащий указанное количество подстрок.

Синтаксис

Split (выражение [, **разделитель** [, **limit** [, **compare**]]] **)**

Часть	Описание
выражение	Необходимые. Строковое выражение, содержащее подстроки и разделители. Если *выражение* представляет собой строку нулевой длины ("" или vbNullString), **Split** возвращает пустой массив, не содержащий элементов и данных. В этом случае возвращаемый массив будет иметь LBound 0 и UBound -1.
ограничитель	Необязательный. Строковый символ, используемый для определения пределов подстроки. Если опустить, символ пробела ("") считается разделителем. Если ***разделителем*** является строка с нулевой длиной, возвращается одноэлементный массив, содержащий всю строку ***выражения*** .
предел	Необязательный. Количество подстрок, подлежащих возврату; -1 указывает, что все подстроки возвращаются.
сравнить	Необязательный. Числовое значение, указывающее, какое сравнение следует использовать при оценке подстрок. См. Раздел «Настройки» для значений.

настройки

Аргумент ***сравнения*** может иметь следующие значения:

постоянная	Значение	Описание
Описание	-1	Выполняет сравнение, используя настройку оператора **сравнения параметров** .
vbBinaryCompare	0	Выполняет двоичное сравнение.
vbTextCompare	1	Выполняет текстовое сравнение.
vbDatabaseCompare	2	Только Microsoft Access. Выполняет сравнение, основанное на информации в вашей базе данных.

пример

В этом примере показано, как Split работает, показывая несколько стилей. В комментариях будет отображаться результирующий набор для каждого из разных вариантов Split. Наконец, показано, как перебирать возвращаемый массив строк.

```
Sub Test

    Dim textArray() as String

    textArray = Split("Tech on the Net")
    'Result: {"Tech", "on", "the", "Net"}

    textArray = Split("172.23.56.4", ".")
    'Result: {"172", "23", "56", "4"}

    textArray = Split("A;B;C;D", ";")
    'Result: {"A", "B", "C", "D"}

    textArray = Split("A;B;C;D", ";", 1)
    'Result: {"A;B;C;D"}

    textArray = Split("A;B;C;D", ";", 2)
    'Result: {"A", "B;C;D"}

    textArray = Split("A;B;C;D", ";", 3)
    'Result: {"A", "B", "C;D"}

    textArray = Split("A;B;C;D", ";", 4)
    'Result: {"A", "B", "C", "D"}

    'You can iterate over the created array
    Dim counter As Long

    For counter = LBound(textArray) To UBound(textArray)
        Debug.Print textArray(counter)
    Next
 End Sub
```

Для ... Далее

Использование переменной итератора в качестве номера индекса является самым быстрым способом для итерации элементов массива:

```
Dim items As Variant
items = Array(0, 1, 2, 3)

Dim index As Integer
For index = LBound(items) To UBound(items)
    'assumes value can be implicitly converted to a String:
    Debug.Print items(index)
Next
```

Вложенные циклы могут использоваться для итерации многомерных массивов:

```
Dim items(0 To 1, 0 To 1) As Integer
items(0, 0) = 0
items(0, 1) = 1
items(1, 0) = 2
items(1, 1) = 3

Dim outer As Integer
Dim inner As Integer
For outer = LBound(items, 1) To UBound(items, 1)
    For inner = LBound(items, 2) To UBound(items, 2)
        'assumes value can be implicitly converted to a String:
        Debug.Print items(outer, inner)
    Next
Next
```

Для каждого ... Далее

А `For Each...Next` цикл также может использоваться для повторения массивов, если производительность не имеет значения:

```
Dim items As Variant
items = Array(0, 1, 2, 3)

Dim item As Variant 'must be variant
For Each item In items
    'assumes value can be implicitly converted to a String:
    Debug.Print item
Next
```

А `For Each` цикла будут выполняться итерация всех измерений от внешнего к внутреннему (в том же порядке, что и элементы, выделенные в памяти), поэтому нет необходимости в вложенных циклах:

```
Dim items(0 To 1, 0 To 1) As Integer
items(0, 0) = 0
items(1, 0) = 1
items(0, 1) = 2
items(1, 1) = 3

Dim item As Variant 'must be Variant
For Each item In items
    'assumes value can be implicitly converted to a String:
    Debug.Print item
Next
```

Обратите внимание, что `For Each` петли лучше всего использовать для итерации объектов `Collection` , если имеет значение производительность.

Все 4 фрагмента выше дают одинаковый результат:

```
 0
 1
 2
 3
```

Динамические массивы (изменение размера массива и динамическая обработка)

Динамические массивы

Добавление и уменьшение переменных в массиве динамически является огромным преимуществом, когда информация, которую вы обрабатываете, не имеет определенного количества переменных.

Добавление значений динамически

Вы можете просто изменить размер массива с помощью `ReDim` , это изменит размер массива, но если вы сохраните информацию, уже сохраненную в массиве, вам понадобится часть `Preserve` .

В приведенном ниже примере мы создаем массив и увеличиваем его на еще одну переменную в каждой итерации, сохраняя значения уже в массиве.

```
Dim Dynamic_array As Variant
' first we set Dynamic_array as variant

For n = 1 To 100

    If IsEmpty(Dynamic_array) Then
        'isempty() will check if we need to add the first value to the array or subsequent
ones
```

```
        ReDim Dynamic_array(0)
        'ReDim Dynamic_array(0) will resize the array to one variable only
        Dynamic_array(0) = n

    Else
        ReDim Preserve Dynamic_array(0 To UBound(Dynamic_array) + 1)
        'in the line above we resize the array from variable 0 to the UBound() = last
variable, plus one effectivelly increeasing the size of the array by one
        Dynamic_array(UBound(Dynamic_array)) = n
        'attribute a value to the last variable of Dynamic_array
    End If

Next
```

Удаление значений динамически

Мы можем использовать ту же логику для уменьшения массива. В этом примере значение «последний» будет удалено из массива.

```
Dim Dynamic_array As Variant
Dynamic_array = Array("first", "middle", "last")

ReDim Preserve Dynamic_array(0 To UBound(Dynamic_array) - 1)
' Resize Preserve while dropping the last value
```

Сброс массива и повторное использование динамически

Мы также можем повторно использовать массивы, которые мы создаем, чтобы не иметь много памяти, что замедлит работу. Это полезно для массивов разных размеров. Один фрагмент кода можно использовать повторно использовать массив в `ReDim` массив обратно `(0)` , приписывать одной переменной в массив и снова свободно увеличивать массив.

В нижеприведенном фрагменте я создаю массив со значениями от 1 до 40, пуст массив и пополняем массив значениями от 40 до 100, все это выполняется динамически.

```
Dim Dynamic_array As Variant

For n = 1 To 100

    If IsEmpty(Dynamic_array) Then
        ReDim Dynamic_array(0)
        Dynamic_array(0) = n

    ElseIf Dynamic_array(0) = "" Then
        'if first variant is empty ( = "") then give it the value of n
        Dynamic_array(0) = n
    Else
        ReDim Preserve Dynamic_array(0 To UBound(Dynamic_array) + 1)
        Dynamic_array(UBound(Dynamic_array)) = n
    End If
    If n = 40 Then
        ReDim Dynamic_array(0)
```

```
        'Resizing the array back to one variable without Preserving,
        'leaving the first value of the array empty
    End If

Next
```

Жесткие массивы (массивы массивов)

Ядра массивов НЕ многомерные массивы

Массивы массивов (Jagged Arrays) не совпадают с многомерными массивами, если вы думаете о них визуально. Многомерные массивы будут выглядеть как Matrices (Rectangular) с определенным количеством элементов по их размерам (внутри массивов), в то время как массив Jagged будет похож на ежегодный календарь с внутренними массивами, имеющими различное количество элементов, например, дни в разные месяцы.

Хотя Jagged Arrays довольно беспорядочны и сложны в использовании из-за их вложенных уровней и не имеют большой безопасности типов, но они очень гибкие, позволяют вам легко манипулировать различными типами данных и не нужно содержать неиспользуемые или пустые элементы.

Создание поврежденного массива

В приведенном ниже примере мы инициализируем зубчатый массив, содержащий два массива один для имен, а другой для чисел, а затем доступ к одному элементу каждого из них

```
Dim OuterArray() As Variant
Dim Names() As Variant
Dim Numbers() As Variant
'arrays are declared variant so we can access attribute any data type to its elements

Names = Array("Person1", "Person2", "Person3")
Numbers = Array("001", "002", "003")

OuterArray = Array(Names, Numbers)
'Directly giving OuterArray an array containing both Names and Numbers arrays inside

Debug.Print OuterArray(0)(1)
Debug.Print OuterArray(1)(1)
'accessing elements inside the jagged by giving the coordenades of the element
```

Динамическое создание и чтение массивов с зубцами

Мы также можем быть более динамичными в нашем приближении, чтобы построить массивы, представьте, что у нас есть личный лист данных клиентов в excel, и мы хотим построить массив для вывода информации о клиенте.

```
   Name -   Phone   -  Email  - Customer Number
Person1 - 153486231 - 1@STACK - 001
Person2 - 153486242 - 2@STACK - 002
Person3 - 153486253 - 3@STACK - 003
Person4 - 153486264 - 4@STACK - 004
Person5 - 153486275 - 5@STACK - 005
```

Мы будем динамически строить массив заголовков и массив Customers, заголовок будет содержать заголовки столбцов, а массив Customers будет содержать информацию каждого клиента / строки в виде массивов.

```
Dim Headers As Variant
' headers array with the top section of the customer data sheet
    For c = 1 To 4
        If IsEmpty(Headers) Then
            ReDim Headers(0)
            Headers(0) = Cells(1, c).Value
        Else
            ReDim Preserve Headers(0 To UBound(Headers) + 1)
            Headers(UBound(Headers)) = Cells(1, c).Value
        End If
    Next

Dim Customers As Variant
'Customers array will contain arrays of customer values
Dim Customer_Values As Variant
'Customer_Values will be an array of the customer in its elements (Name-Phone-Email-CustNum)

    For r = 2 To 6
    'iterate through the customers/rows
        For c = 1 To 4
        'iterate through the values/columns

            'build array containing customer values
            If IsEmpty(Customer_Values) Then
                ReDim Customer_Values(0)
                Customer_Values(0) = Cells(r, c).Value
            ElseIf Customer_Values(0) = "" Then
                Customer_Values(0) = Cells(r, c).Value
            Else
                ReDim Preserve Customer_Values(0 To UBound(Customer_Values) + 1)
                Customer_Values(UBound(Customer_Values)) = Cells(r, c).Value
            End If
        Next

        'add customer_values array to Customers Array
        If IsEmpty(Customers) Then
            ReDim Customers(0)
            Customers(0) = Customer_Values
        Else
            ReDim Preserve Customers(0 To UBound(Customers) + 1)
            Customers(UBound(Customers)) = Customer_Values
        End If

        'reset Custumer_Values to rebuild a new array if needed
        ReDim Customer_Values(0)
    Next

    Dim Main_Array(0 To 1) As Variant
```

```
    'main array will contain both the Headers and Customers

    Main_Array(0) = Headers
    Main_Array(1) = Customers
```

To better understand the way to Dynamically construct a one dimensional array please check Dynamic Arrays (Array Resizing and Dynamic Handling) on the Arrays documentation.

Результатом приведенного выше фрагмента является массив с чередованием с двумя массивами, один из тех массивов с 4 элементами, 2 уровня отступа, а другой сам по себе является другим массивом Jagged, содержащим 5 массивов из 4 элементов каждый и 3 уровня отступа, см. Ниже структуру:

```
Main_Array(0) - Headers - Array("Name","Phone","Email","Customer Number")
          (1) - Customers(0) - Array("Person1",153486231,"1@STACK",001)
                Customers(1) - Array("Person2",153486242,"2@STACK",002)
                ...
                Customers(4) - Array("Person5",153486275,"5@STACK",005)
```

Чтобы получить доступ к информации, которую вы должны иметь в виду о структуре созданного массива Jagged Array, в приведенном выше примере вы можете видеть, что `Main Array` содержит массив `Headers` и массив массивов (`Customers`), следовательно, с различными способами доступ к элементам.

Теперь мы прочитаем информацию о `Main Array` и распечатаем каждую из данных Клиентов в виде `Info Type: Info` .

```
For n = 0 To UBound(Main_Array(1))
    'n to iterate from fisrt to last array in Main_Array(1)

    For j = 0 To UBound(Main_Array(1)(n))
        'j will iterate from first to last element in each array of Main_Array(1)

        Debug.Print Main_Array(0)(j) & ": " & Main_Array(1)(n)(j)
        'print Main_Array(0)(j) which is the header and Main_Array(0)(n)(j) which is the
element in the customer array
        'we can call the header with j as the header array has the same structure as the
customer array
    Next
Next
```

ЗАПОМНИТЕ, чтобы отслеживать структуру вашего Jagged Array, в приведенном выше примере для доступа к имени клиента, `Main_Array -> Customers -> CustomerNumber -> Name` к `Main_Array -> Customers -> CustomerNumber -> Name` который состоит из трех уровней, для возврата `"Person4"` вам понадобится расположение клиентов в Main_Array, затем местоположение клиента 4 в массиве Jagged Customers и, наконец, местоположение `Main_Array(1)(3)(0)` элемента в этом случае `Main_Array(1)(3)(0)` который является `Main_Array(Customers)(CustomerNumber)(Name)` .

Многомерные массивы

Как видно из названия, многомерные массивы представляют собой массивы, которые содержат более одного измерения, обычно два или три, но могут иметь до 32 измерений.

Массив массива работает как матрица с различными уровнями, например, сравнение между одним, двумя и тремя измерениями.

Одно измерение - ваш типичный массив, он выглядит как список элементов.

```
Dim 1D(3) as Variant

*1D - Visually*
(0)
(1)
(2)
```

Два измерения будут выглядеть как сетка Sudoku или лист Excel, при инициализации массива вы определяете, сколько строк и столбцов будет иметь массив.

```
Dim 2D(3,3) as Variant
'this would result in a 3x3 grid

*2D - Visually*
(0,0) (0,1) (0,2)
(1,0) (1,1) (1,2)
(2,0) (2,1) (2,2)
```

Три измерения начнут выглядеть как кубик Рубика, при инициализации массива вы будете определять строки и столбцы, а также уровни / глубины, которые будет иметь массив.

```
Dim 3D(3,3,2) as Variant
'this would result in a 3x3x3 grid

*3D - Visually*
       1st layer                 2nd layer                   3rd layer
         front                     middle                      back
(0,0,0) (0,0,1) (0,0,2) ¦ (1,0,0) (1,0,1) (1,0,2) ¦ (2,0,0) (2,0,1) (2,0,2)
(0,1,0) (0,1,1) (0,1,2) ¦ (1,1,0) (1,1,1) (1,1,2) ¦ (2,1,0) (2,1,1) (2,1,2)
(0,2,0) (0,2,1) (0,2,2) ¦ (1,2,0) (1,2,1) (1,2,2) ¦ (2,2,0) (2,2,1) (2,2,2)
```

Дальнейшие измерения можно рассматривать как умножение 3D, поэтому 4D (1,3,3,3) будет представлять собой два боковых боковых 3D-массива.

Двухмерный массив

Создание

Пример ниже будет компиляцией списка сотрудников, каждый сотрудник будет иметь набор информации в списке (имя, фамилия, адрес, электронная почта, телефон ...), пример, по существу, будет храниться в массиве (сотрудник, информация), являющееся (0,0), является первым именем первого сотрудника.

```
Dim Bosses As Variant
'set bosses as Variant, so we can input any data type we want

Bosses = [{"Jonh","Snow","President";"Ygritte","Wild","Vice-President"}]
'initialise a 2D array directly by filling it with information, the redult wil be a array(1,2)
size 2x3 = 6 elements

Dim Employees As Variant
'initialize your Employees array as variant
'initialize and ReDim the Employee array so it is a dynamic array instead of a static one,
hence treated differently by the VBA Compiler
ReDim Employees(100, 5)
'declaring an 2D array that can store 100 employees with 6 elements of information each, but
starts empty
'the array size is 101 x 6 and contains 606 elements

For employee = 0 To UBound(Employees, 1)
'for each employee/row in the array, UBound for 2D arrays, which will get the last element on
the array
'needs two parameters 1st the array you which to check and 2nd the dimension, in this case 1 =
employee and 2 = information
    For information_e = 0 To UBound(Employees, 2)
    'for each information element/column in the array

        Employees(employee, information_e) = InformationNeeded ' InformationNeeded would be
the data to fill the array
        'iterating the full array will allow for direct attribution of information into the
element coordinates
    Next
Next
```

Изменение размера

Изменение размера или `ReDim Preserve` Multi-Array, как и норма для массива One-Dimension, приведет к ошибке, вместо этого информация должна быть перенесена в массив Temporary с тем же размером, что и оригинал, плюс число добавляемых строк / столбцов. В приведенном ниже примере мы увидим, как инициализировать Temp Array, передать информацию из исходного массива, заполнить оставшиеся пустые элементы и заменить массив temp на исходный массив.

```
Dim TempEmp As Variant
'initialise your temp array as variant
ReDim TempEmp(UBound(Employees, 1) + 1, UBound(Employees, 2))
'ReDim/Resize Temp array as a 2D array with size UBound(Employees)+1 = (last element in
Employees 1st dimension) + 1,
'the 2nd dimension remains the same as the original array. we effectively add 1 row in the
Employee array
```

```
'transfer
For emp = LBound(Employees, 1) To UBound(Employees, 1)
    For info = LBound(Employees, 2) To UBound(Employees, 2)
        'to transfer Employees into TempEmp we iterate both arrays and fill TempEmp with the
corresponding element value in Employees
        TempEmp(emp, info) = Employees(emp, info)

    Next
Next

'fill remaining
'after the transfers the Temp array still has unused elements at the end, being that it was
increased
'to fill the remaining elements iterate from the last "row" with values to the last row in the
array
'in this case the last row in Temp will be the size of the Employees array rows + 1, as the
last row of Employees array is already filled in the TempArray

For emp = UBound(Employees, 1) + 1 To UBound(TempEmp, 1)
    For info = LBound(TempEmp, 2) To UBound(TempEmp, 2)

        TempEmp(emp, info) = InformationNeeded & "NewRow"

    Next
Next

'erase Employees, attribute Temp array to Employees and erase Temp array
Erase Employees
Employees = TempEmp
Erase TempEmp
```

Изменение значений элементов

Изменение / изменение значений в определенном элементе может быть выполнено путем простого вызова координаты для изменения и предоставления ей нового значения:

```
Employees(0, 0) = "NewValue"
```

В качестве альтернативы, итерация по координатам использует условия для соответствия значениям, соответствующим требуемым параметрам:

```
For emp = 0 To UBound(Employees)
    If Employees(emp, 0) = "Gloria" And Employees(emp, 1) = "Stephan" Then
    'if value found
        Employees(emp, 1) = "Married, Last Name Change"
        Exit For
        'don't iterate through a full array unless necessary
    End If
Next
```

чтение

Доступ к элементам в массиве может выполняться с помощью вложенного цикла (итерация каждого элемента), цикла и координат (итерации строк и доступа к столбцам напрямую)

или прямого доступа к обеим координатам.

```
'nested loop, will iterate through all elements
For emp = LBound(Employees, 1) To UBound(Employees, 1)
    For info = LBound(Employees, 2) To UBound(Employees, 2)
        Debug.Print Employees(emp, info)
    Next
Next

'loop and coordinate, iteration through all rows and in each row accessing all columns
directly
For emp = LBound(Employees, 1) To UBound(Employees, 1)
    Debug.Print Employees(emp, 0)
    Debug.Print Employees(emp, 1)
    Debug.Print Employees(emp, 2)
    Debug.Print Employees(emp, 3)
    Debug.Print Employees(emp, 4)
    Debug.Print Employees(emp, 5)
Next

'directly accessing element with coordinates
Debug.Print Employees(5, 5)
```

***Помните** , всегда удобно хранить карту массива при использовании многомерных массивов, они могут легко стать путаницей.*

Трехмерный массив

Для 3D-массива мы будем использовать ту же предпосылку, что и 2D-массив, с добавлением не только хранения Employee и Information, но и построения, в котором они работают.

В трехмерном массиве будут присутствовать сотрудники (могут рассматриваться как строки), информация (столбцы) и здание, которые можно рассматривать как разные листы на документе excel, они имеют одинаковый размер между ними, но каждый лист имеет различный набор информации в своих ячейках / элементах. 3D-массив будет содержать ***n*** число 2D-массивов.

Создание

3D-массив нуждается в трех координатах для инициализации `Dim 3Darray(2,5,5) As Variant` первой координатой массива будет количество строений / таблиц (разные наборы строк и столбцов), вторая координата будет определять строки и третью Столбцы. В результате `Dim` выше будет создан трехмерный массив с 108 элементами (`3*6*6`), эффективно имеющий 3 разных набора 2D-массивов.

```
Dim ThreeDArray As Variant
'initialise your ThreeDArray array as variant
ReDim ThreeDArray(1, 50, 5)
```

```
'declaring an 3D array that can store two sets of 51 employees with 6 elements of information
each, but starts empty
'the array size is 2 x 51 x 6 and contains 612 elements

For building = 0 To UBound(ThreeDArray, 1)
    'for each building/set in the array
    For employee = 0 To UBound(ThreeDArray, 2)
    'for each employee/row in the array
        For information_e = 0 To UBound(ThreeDArray, 3)
        'for each information element/column in the array

            ThreeDArray(building, employee, information_e) = InformationNeeded '
InformationNeeded would be the data to fill the array
        'iterating the full array will allow for direct attribution of information into the
element coordinates
        Next
    Next
Next
```

Изменение размера

Изменение размера 3D-массива аналогично изменению размера 2D, сначала создайте временный массив с тем же размером оригинала, добавляя его в координату параметра для увеличения, первая координата увеличит количество наборов в массиве, второе и третьи координаты увеличат количество строк или столбцов в каждом наборе.

Пример ниже увеличивает количество строк в каждом наборе на единицу и заполняет те недавно добавленные элементы новой информацией.

```
Dim TempEmp As Variant
'initialise your temp array as variant
ReDim TempEmp(UBound(ThreeDArray, 1), UBound(ThreeDArray, 2) + 1, UBound(ThreeDArray, 3))
'ReDim/Resize Temp array as a 3D array with size UBound(ThreeDArray)+1 = (last element in
Employees 2nd dimension) + 1,
'the other dimension remains the same as the original array. we effectively add 1 row in the
for each set of the 3D array

'transfer
For building = LBound(ThreeDArray, 1) To UBound(ThreeDArray, 1)
    For emp = LBound(ThreeDArray, 2) To UBound(ThreeDArray, 2)
        For info = LBound(ThreeDArray, 3) To UBound(ThreeDArray, 3)
            'to transfer ThreeDArray into TempEmp by iterating all sets in the 3D array and
fill TempEmp with the corresponding element value in each set of each row
            TempEmp(building, emp, info) = ThreeDArray(building, emp, info)

        Next
    Next
Next

'fill remaining
'to fill the remaining elements we need to iterate from the last "row" with values to the last
row in the array in each set, remember that the first empty element is the original array
Ubound() plus 1
For building = LBound(TempEmp, 1) To UBound(TempEmp, 1)
    For emp = UBound(ThreeDArray, 2) + 1 To UBound(TempEmp, 2)
        For info = LBound(TempEmp, 3) To UBound(TempEmp, 3)
```

```
            TempEmp(building, emp, info) = InformationNeeded & "NewRow"

        Next
    Next
Next

'erase Employees, attribute Temp array to Employees and erase Temp array
Erase ThreeDArray
ThreeDArray = TempEmp
Erase TempEmp
```

Изменение значений элементов и чтение

Чтение и изменение элементов в 3D-массиве может быть выполнено аналогично тому, как мы делаем 2D-массив, просто отрегулируйте дополнительный уровень в петлях и координатах.

```
Do
' using Do ... While for early exit
    For building = 0 To UBound(ThreeDArray, 1)
        For emp = 0 To UBound(ThreeDArray, 2)
            If ThreeDArray(building, emp, 0) = "Gloria" And ThreeDArray(building, emp, 1) =
"Stephan" Then
            'if value found
                ThreeDArray(building, emp, 1) = "Married, Last Name Change"
                Exit Do
                'don't iterate through all the array unless necessary
            End If
        Next
    Next
Loop While False

'nested loop, will iterate through all elements
For building = LBound(ThreeDArray, 1) To UBound(ThreeDArray, 1)
    For emp = LBound(ThreeDArray, 2) To UBound(ThreeDArray, 2)
        For info = LBound(ThreeDArray, 3) To UBound(ThreeDArray, 3)
            Debug.Print ThreeDArray(building, emp, info)
        Next
    Next
Next

'loop and coordinate, will iterate through all set of rows and ask for the row plus the value
we choose for the columns
For building = LBound(ThreeDArray, 1) To UBound(ThreeDArray, 1)
    For emp = LBound(ThreeDArray, 2) To UBound(ThreeDArray, 2)
        Debug.Print ThreeDArray(building, emp, 0)
        Debug.Print ThreeDArray(building, emp, 1)
        Debug.Print ThreeDArray(building, emp, 2)
        Debug.Print ThreeDArray(building, emp, 3)
        Debug.Print ThreeDArray(building, emp, 4)
        Debug.Print ThreeDArray(building, emp, 5)
    Next
Next

'directly accessing element with coordinates
Debug.Print Employees(0, 5, 5)
```

Прочитайте Массивы онлайн: https://riptutorial.com/ru/vba/topic/3064/массивы

глава 17: Назначение строк с повторяющимися символами

замечания

Иногда вам нужно назначить строковую переменную с конкретным символом, который повторяется определенное количество раз. VBA обеспечивает две основные функции для этой цели:

- `String` / `String$`
- `Space` / `Space$` .

Examples

Используйте функцию String, чтобы назначить строку с n повторными символами

```
Dim lineOfHyphens As String
'Assign a string with 80 repeated hyphens
lineOfHyphens = String$(80, "-")
```

Используйте функции String и Space для назначения строки n-символа

```
Dim stringOfSpaces As String

'Assign a string with 255 repeated spaces using Space$
stringOfSpaces = Space$(255)

'Assign a string with 255 repeated spaces using String$
stringOfSpaces = String$(255, " ")
```

Прочитайте Назначение строк с повторяющимися символами онлайн: https://riptutorial.com/ru/vba/topic/3581/назначение-строк-с-повторяющимися-символами

глава 18: Нелатинские символы

Вступление

VBA может читать и записывать строки на любом языке или скрипте с использованием Unicode . Тем не менее, существуют более строгие правила для токенов идентификатора .

Examples

Нелатинский текст в коде VBA

В ячейке A1 для электронных таблиц мы имеем следующую арабскую панграмму:

صف خلق خود كمثل الشمس إذ بزغت - يحظى الضجيع بها نجلاء معطار

VBA предоставляет функции `AscW` и `ChrW` для работы с многобайтовыми кодами символов. Мы также можем использовать `Byte` массивы для непосредственного управления строковой переменной:

```
Sub NonLatinStrings()

Dim rng As Range
Set rng = Range("A1")
Do Until rng = ""
    Dim MyString As String
    MyString = rng.Value

    ' AscW functions
    Dim char As String
    char = AscW(Left(MyString, 1))
    Debug.Print "First char (ChrW): " & char
    Debug.Print "First char (binary): " & BinaryFormat(char, 12)

    ' ChrW functions
    Dim uString As String
    uString = ChrW(char)
    Debug.Print "String value (text): " & uString        ' Fails! Appears as '?'
    Debug.Print "String value (AscW): " & AscW(uString)

    ' Using a Byte string
    Dim StringAsByt() As Byte
    StringAsByt = MyString
    Dim i As Long
    For i = 0 To 1 Step 2
        Debug.Print "Byte values (in decimal): " & _
            StringAsByt(i) & "|" & StringAsByt(i + 1)
        Debug.Print "Byte values (binary): " & _
            BinaryFormat(StringAsByt(i)) & "|" & BinaryFormat(StringAsByt(i + 1))
    Next i
    Debug.Print ""
```

```
    ' Printing the entire string to the immediate window fails (all '?'s)
    Debug.Print "Whole String" & vbNewLine & rng.Value
    Set rng = rng.Offset(1)
Loop

End Sub
```

Это приводит к следующему выводу для арабского письма :

> Первый символ (ChrW): 1589
> Первый символ (двоичный): 00011000110101
> Строковое значение (текст):?
> Строковое значение (AscW): 1589
> Значения байтов (в десятичной форме): 53 | 6
> Значения байтов (двоичные): 00110101 | 00000110
>
> Целая строка
> *??? ????? ????? ??????? ??????? ??? ??????? - ????? ???????? ???? ??????? ???????*

Обратите внимание, что VBA не может печатать нелатинский текст в непосредственном окне, даже если строковые функции работают правильно. Это ограничение IDE, а не языка.

Нелатинские идентификаторы и языковой охват

Идентификаторы VBA (имена переменных и функций) могут использовать латинский скрипт и также могут использовать японские , корейские , упрощенные китайские и традиционные китайские скрипты.

Расширенный латинский скрипт имеет полный охват для многих языков:
Английский, Французский, Испанский, Немецкий, Итальянский, Бретонский, Каталонский, Датский, Эстонский, Финский, Исландский, Индонезийский, Ирландский, Лойбанский, Мапудунгун, Норвежский, Португальский, Шотландский гэльский, Шведский, Тагалог

Некоторые языки покрываются лишь частично:
Азербайджанский, хорватский, чешский, эсперанто, венгерский, латышский, литовский, польский, румынский, сербский, словацкий, словенский, турецкий, йоруба, валлийский

Некоторые языки практически не имеют охвата:
Арабский, болгарский, чероки, дзонгха, греческий, хинди, македонский, малаялам, монгольский, русский, санскритский, тайский, тибетский, урдуский, уйгурский

Все допустимые объявления переменных действительны:

```
Dim Yec'hed As String 'Breton
Dim «Dóna» As String 'Catalan
```

```
Dim fræk As String 'Danish
Dim tšellomängija As String 'Estonian
Dim Törkylempijävongahdus As String 'Finnish
Dim j’examine As String 'French
Dim Paß As String 'German
Dim þjófum As String 'Icelandic
Dim hÓighe As String 'Irish
Dim sofybakni As String 'Lojban (.o’i does not work)
Dim ñizol As String 'Mapudungun
Dim Vår As String 'Norwegian
Dim «brações» As String 'Portuguese
Dim d’fhàg As String 'Scottish Gaelic
```

Обратите внимание, что в VBA IDE один апостроф внутри имени переменной не превращает строку в комментарий (как это делается при переполнении стека).

Кроме того, на языках, которые используют два угла для указания цитаты «», разрешено использовать те, которые в именах переменных desipte используют тот факт, что котировки типа «» не являются.

Прочитайте Нелатинские символы онлайн: https://riptutorial.com/ru/vba/topic/10555/нелатинские-символы

глава 19: Обработка ошибок

Examples

Избегание условий ошибки

Когда возникает ошибка времени выполнения, хороший код должен ее обрабатывать. Лучшей стратегией обработки ошибок является запись кода, который проверяет условия ошибки и просто избегает выполнения кода, который приводит к ошибке выполнения.

Одним из ключевых элементов сокращения ошибок во время выполнения является запись небольших процедур, которые *делают одно* . Чем меньше процедур процедур приходится терпеть неудачу, тем проще код в целом - отлаживать.

Избежать ошибки времени выполнения 91 - Объект или С заблокированной переменной блока:

Эта ошибка будет повышена, если объект используется до назначения ссылки. Возможно, у вас есть процедура, которая получает параметр объекта:

```
Private Sub DoSomething(ByVal target As Worksheet)
    Debug.Print target.Name
End Sub
```

Если `target` не назначена ссылка, приведенный выше код вызовет ошибку, которую легко избежать, проверяя, содержит ли объект фактическую ссылку на объект:

```
Private Sub DoSomething(ByVal target As Worksheet)
    If target Is Nothing Then Exit Sub
    Debug.Print target.Name
End Sub
```

Если `target` назначению не присвоена ссылка, то непризнанная ссылка никогда не используется, и ошибка не возникает.

Этот способ раннего выхода из процедуры, когда один или несколько параметров недопустимы, называется предложением *охраны* .

Избегайте ошибки времени выполнения 9 - Подкласс вне диапазона:

Эта ошибка возникает при доступе к массиву за пределами его границ.

```
Private Sub DoSomething(ByVal index As Integer)
    Debug.Print ActiveWorkbook.Worksheets(index)
```

```
End Sub
```

Учитывая, что индекс больше, чем количество листов в `ActiveWorkbook` , приведенный выше код вызовет ошибку времени выполнения. Простое предложение охраны может избежать этого:

```
Private Sub DoSomething(ByVal index As Integer)
    If index > ActiveWorkbook.Worksheets.Count Or index <= 0 Then Exit Sub
    Debug.Print ActiveWorkbook.Worksheets(index)
End Sub
```

Большинство ошибок времени выполнения можно избежать, тщательно проверив значения, которые мы используем, *прежде чем* мы их используем, и разветвляемся на другом пути выполнения, соответственно, используя простой оператор `If` - в сторожевых предложениях, который не делает предположений и не проверяет параметры процедуры, или даже в тело более крупных процедур.

Оператор Error

Даже с *защитными пунктами,* один не может реально *всегда* учитывать все возможные ошибки , которые могут быть подняты в теле процедуры. Оператор `On Error GoTo` инструктирует VBA перейти к *метке линии* и ввести «режим обработки ошибок» всякий раз, когда во время выполнения происходит непредвиденная ошибка. После обработки ошибки, код может *возобновить* обратно в «нормальное» исполнение с помощью `Resume` ключевое слово.

Линейные метки обозначают *подпрограммы* : потому что подпрограммы исходят из устаревшего кода BASIC и используют `GoSub GoTo` и `GoSub` и `Return` чтобы вернуться к «основной» процедуре, довольно легко написать жесткий *код спагетти,* если все не строго структурировано , По этой причине лучше всего:

- процедура имеет **одну и только одну** подпрограмму обработки ошибок
- подпрограмма обработки ошибок **работает только в состоянии ошибки**

Это означает, что процедура, которая обрабатывает его ошибки, должна быть структурирована следующим образом:

```
Private Sub DoSomething()
    On Error GoTo CleanFail

    'procedure code here

CleanExit:
    'cleanup code here
    Exit Sub

CleanFail:
    'error-handling code here
```

```
    Resume CleanExit
End Sub
```

Стратегии обработки ошибок

Иногда вы хотите обрабатывать разные ошибки с помощью разных действий. В этом случае вы будете проверять глобальный объект `Err` , который будет содержать информацию об ошибке, которая была поднята, и действовать соответственно:

```
CleanExit:
    Exit Sub

CleanFail:
    Select Case Err.Number
        Case 9
            MsgBox "Specified number doesn't exist. Please try again.", vbExclamation
            Resume
        Case 91
            'woah there, this shouldn't be happening.
            Stop 'execution will break here
            Resume 'hit F8 to jump to the line that raised the error
        Case Else
            MsgBox "An unexpected error has occurred:" & vbNewLine & Err.Description,
vbCritical
            Resume CleanExit
    End Select
End Sub
```

В качестве общего руководства рассмотрите возможность включения обработки ошибок для всей подпрограммы или функции и обработайте все ошибки, которые могут возникнуть в пределах ее области действия. Если вам нужно обрабатывать ошибки только в секции небольшого сечения кода - включить и выключить обработку ошибок на одном уровне:

```
Private Sub DoSomething(CheckValue as Long)

    If CheckValue = 0 Then
        On Error GoTo ErrorHandler   ' turn error handling on
        ' code that may result in error
        On Error GoTo 0              ' turn error handling off - same level
    End If

CleanExit:
    Exit Sub

ErrorHandler:
    ' error handling code here
    ' do not turn off error handling here
    Resume

End Sub
```

Номера строк

VBA поддерживает номера строк в стиле legacy (например, QBASIC). Скрытое свойство `Erl` можно использовать для идентификации номера строки, которая вызвала последнюю ошибку. Если вы не используете номера строк, `Erl` только вернет 0.

```
Sub DoSomething()
10 On Error GoTo 50
20 Debug.Print 42 / 0
30 Exit Sub
40
50 Debug.Print "Error raised on line " & Erl ' returns 20
End Sub
```

Если вы используете номера строк, но не последовательно, а затем `Erl` возвращает *номер последней строки перед командой, вызвавшей ошибку.*

```
Sub DoSomething()
10 On Error GoTo 50
   Debug.Print 42 / 0
30 Exit Sub

50 Debug.Print "Error raised on line " & Erl 'returns 10
End Sub
```

Имейте в виду, что `Erl` также имеет только `Integer` точность и будет бесшумно переполняться. Это означает, что номера строк за пределами целочисленного диапазона дадут неверные результаты:

```
Sub DoSomething()
99997 On Error GoTo 99999
99998 Debug.Print 42 / 0
99999
      Debug.Print Erl   'Prints 34462
End Sub
```

Номер строки не так актуален, как утверждение, вызвавшее ошибку, и строки нумерации быстро становятся утомительными и не совсем удобны в обслуживании.

Резюме ключевого слова

Подпрограмма обработки ошибок будет либо:

- выполняются до конца процедуры, и в этом случае выполнение возобновляется в процедуре вызова.
- или используйте ключевое слово `Resume` для *возобновления* выполнения внутри той же процедуры.

Ключевое слово `Resume` должно использоваться только в подпрограмме обработки ошибок, потому что если VBA встречает `Resume` не находясь в состоянии ошибки, возникает ошибка времени выполнения 20 «Возобновить без ошибок».

Существует несколько способов, по которым подпрограмма обработки ошибок может использовать ключевое слово `Resume` :

- `Resume` используется отдельно, выполнение продолжается **в инструкции, вызвавшей ошибку** . Если ошибка на *самом деле* не обрабатывается , прежде чем делать это, то та же ошибка будет поднят снова, и выполнение может войти в бесконечный цикл.
- `Resume Next` продолжает выполнение **инструкции сразу после** инструкции, вызвавшей ошибку. Если ошибка на *самом деле* не обрабатывается , прежде чем делать это, то выполнение разрешается продолжать с потенциально недействительными данными, которые могут привести к логическим ошибкам и неожиданному поведению.
- `Resume [line label]` продолжает выполнение **на указанной метке строки** (или номер строки, если вы используете номера строк в стиле устаревшего стиля). Обычно это позволяет выполнить некоторый код очистки до того, как будет чисто выйти из процедуры, например, чтобы закрыть соединение с базой данных, прежде чем вернуться к вызывающему.

Вкл.

Сам оператор `On Error` может использовать ключевое слово `Resume` чтобы проинструктировать среду выполнения VBA для эффективного **игнорирования всех ошибок** .

> *Если ошибка не **выполняется** до этого, то выполнение разрешено продолжать с потенциально недействительными данными, что может привести к **логическим ошибкам и неожиданному поведению** .*

Вышеупомянутый акцент не может быть особо подчеркнут. ***On Error Resume Next* эффективно игнорирует все ошибки и выталкивает их под ковер** . Программа, которая взрывается с ошибкой во время выполнения с учетом недопустимого ввода, - это более эффективная программа, чем программа, которая работает с неизвестными / непреднамеренными данными - будь то только потому, что ошибка намного легче идентифицируется. `On Error Resume Next` можно легко **скрыть ошибки** .

Оператор `On Error` является областью действия процедур - поэтому в данной процедуре *обычно* должен быть только **один** , такой оператор `On Error` .

Однако *иногда* не удается избежать ошибки, и переключение на подпрограмму обработки ошибок только на `Resume Next` просто не кажется правильным. В этом конкретном случае утверждение с известным до невозможности может быть **обернуто** между двумя `On Error` :

```
On Error Resume Next
[possibly-failing statement]
Err.Clear 'resets current error
On Error GoTo 0
```

Команда `On Error GoTo 0` сбрасывает обработку ошибок в текущей процедуре, так что любая дополнительная инструкция, вызывающая ошибку времени выполнения *, будет необработанной внутри этой процедуры* и вместо этого будет переходить в стек вызовов до тех пор, пока она не будет захвачена активным обработчиком ошибок. Если в стеке вызовов нет активного обработчика ошибок, он будет рассматриваться как необработанное исключение.

```
Public Sub Caller()
    On Error GoTo Handler

    Callee

    Exit Sub
Handler:
    Debug.Print "Error " & Err.Number & " in Caller."
End Sub

Public Sub Callee()
    On Error GoTo Handler

    Err.Raise 1     'This will be handled by the Callee handler.
    On Error GoTo 0 'After this statement, errors are passed up the stack.
    Err.Raise 2     'This will be handled by the Caller handler.

    Exit Sub
Handler:
    Debug.Print "Error " & Err.Number & " in Callee."
    Resume Next
End Sub
```

Пользовательские ошибки

Часто при написании специализированного класса вы хотите, чтобы он поднимал свои собственные конкретные ошибки, и вам понадобится чистый способ для кода пользователя / вызова для обработки этих пользовательских ошибок. Оптимальным способом достижения этого является определение специального типа `Enum` :

```
Option Explicit
Public Enum FoobarError
    Err_FooWasNotBarred = vbObjectError + 1024
    Err_BarNotInitialized
    Err_SomethingElseHappened
End Enum
```

Используя встроенную константу `vbObjectError` пользовательские коды ошибок не перекрываются с зарезервированными / существующими кодами ошибок. Необходимо явно указать только первое значение перечисления, поскольку базовое значение каждого

члена `Enum` `1` больше, чем предыдущий элемент, поэтому базовое значение `Err_BarNotInitialized` неявно является `vbObjectError + 1025` .

Повышение собственных ошибок времени выполнения

Ошибка выполнения может быть повышена с `Err.Raise` оператора `Err.Raise` , поэтому пользовательская ошибка `Err_FooWasNotBarred` может быть повышена следующим образом:

```
Err.Raise Err_FooWasNotBarred
```

Метод `Err.Raise` также может принимать пользовательские параметры `Description` и `Source` - по этой причине рекомендуется также определять константы для хранения каждого пользовательского описания ошибки:

```
Private Const Msg_FooWasNotBarred As String = "The foo was not barred."
Private Const Msg_BarNotInitialized As String = "The bar was not initialized."
```

А затем создайте выделенный частный метод для повышения каждой ошибки:

```
Private Sub OnFooWasNotBarredError(ByVal source As String)
    Err.Raise Err_FooWasNotBarred, source, Msg_FooWasNotBarred
End Sub

Private Sub OnBarNotInitializedError(ByVal source As String)
    Err.Raise Err_BarNotInitialized, source, Msg_BarNotInitialized
End Sub
```

После этого реализация класса может просто вызвать эти специализированные процедуры для повышения ошибки:

```
Public Sub DoSomething()
    'raises the custom 'BarNotInitialized' error with "DoSomething" as the source:
    If Me.Bar Is Nothing Then OnBarNotInitializedError "DoSomething"
    '...
End Sub
```

Клиентский код может обрабатывать `Err_BarNotInitialized` как и любую другую ошибку, внутри своей собственной подпрограммы обработки ошибок.

Примечание: наследие `Error` ключевое слово также может быть использован вместо `Err.Raise` , но это устаревшее / осуждается.

Прочитайте Обработка ошибок онлайн: https://riptutorial.com/ru/vba/topic/3211/обработка-ошибок

глава 20: Объект Scripting.Dictionary

замечания

Вы должны добавить Runtime Microsoft Scripting в проект VBA с помощью команды «Инструменты» → «Ссылки» VBE, чтобы реализовать раннее связывание объекта Scripting Dictionary. Эта библиографическая ссылка приводится в проекте; ему не нужно повторно ссылаться, когда проект VBA распространяется и запускается на другом компьютере.

Examples

Свойства и методы

Объект Scripting Dictionary хранит информацию в парах Key / Item. Ключи должны быть уникальными, а не массивом, но ассоциированные элементы могут быть повторены (их уникальность удерживается ключом компаньона) и может быть любого типа варианта или объекта.

Словарь можно рассматривать как базу данных с двумя полями в базе данных с основным уникальным индексом в первом «поле» (*Key*). Этот уникальный индекс в свойстве Keys позволяет очень быстро «искать» для получения значения связанного с ним элемента.

свойства

название	читай пиши	тип	описание
CompareMode	*читай пиши*	Константа CompareMode	Установка CompareMode может выполняться только в пустом словаре. Принятые значения: 0 (vbBinaryCompare), 1 (vbTextCompare), 2 (vbDatabaseCompare).
подсчитывать	*только для чтения*	unsigned long integer	Однозначное количество пар ключей / элементов в объекте словаря сценариев.
ключ	*читай пиши*	вариант без массива	Каждый отдельный уникальный ключ в словаре.
Элемент (*Ключ*)	*читай пиши*	любой вариант	Свойство по умолчанию. Каждый отдельный элемент, связанный с ключом в словаре. Обратите внимание, что попытка

название	читай пиши	тип	описание
			получить элемент с ключом, который не существует в словаре, *неявно добавит* переданный ключ.

методы

название	описание
Добавить (*ключ* , *элемент*)	Добавляет новый ключ и элемент в словарь. Новый ключ не должен существовать в текущей коллекции клавиш в словаре, но элемент можно повторить среди множества уникальных ключей.
Существует (*ключ*)	Boolean test, чтобы определить, существует ли ключ в словаре.
Ключи	Возвращает массив или набор уникальных ключей.
Предметы	Возвращает массив или набор связанных элементов.
Удалить (*ключ*)	Удаляет отдельный ключ словаря и связанный с ним элемент.
Удалить все	Удаляет все ключи и элементы словарного объекта.

Образец кода

```
'Populate, enumerate, locate and remove entries in a dictionary that was created
'with late binding
Sub iterateDictionaryLate()
    Dim k As Variant, dict As Object

    Set dict = CreateObject("Scripting.Dictionary")
    dict.CompareMode = vbTextCompare          'non-case sensitive compare model

    'populate the dictionary
    dict.Add Key:="Red", Item:="Balloon"
    dict.Add Key:="Green", Item:="Balloon"
    dict.Add Key:="Blue", Item:="Balloon"

    'iterate through the keys
    For Each k In dict.Keys
        Debug.Print k & " - " & dict.Item(k)
    Next k

    'locate the Item for Green
    Debug.Print dict.Item("Green")

    'remove key/item pairs from the dictionary
    dict.Remove "blue"      'remove individual key/item pair by key
```

```
    dict.RemoveAll          'remove all remaining key/item pairs

End Sub

'Populate, enumerate, locate and remove entries in a dictionary that was created
'with early binding (see Remarks)
Sub iterateDictionaryEarly()
    Dim d As Long, k As Variant
    Dim dict As New Scripting.Dictionary

    dict.CompareMode = vbTextCompare          'non-case sensitive compare model

    'populate the dictionary
    dict.Add Key:="Red", Item:="Balloon"
    dict.Add Key:="Green", Item:="Balloon"
    dict.Add Key:="Blue", Item:="Balloon"
    dict.Add Key:="White", Item:="Balloon"

    'iterate through the keys
    For Each k In dict.Keys
        Debug.Print k & " - " & dict.Item(k)
    Next k

    'iterate through the keys by the count
    For d = 0 To dict.Count - 1
        Debug.Print dict.Keys(d) & " - " & dict.Items(d)
    Next d

    'iterate through the keys by the boundaries of the keys collection
    For d = LBound(dict.Keys) To UBound(dict.Keys)
        Debug.Print dict.Keys(d) & " - " & dict.Items(d)
    Next d

    'locate the Item for Green
    Debug.Print dict.Item("Green")
    'locate the Item for the first key
    Debug.Print dict.Item(dict.Keys(0))
    'locate the Item for the last key
    Debug.Print dict.Item(dict.Keys(UBound(dict.Keys)))

    'remove key/item pairs from the dictionary
    dict.Remove "blue"                          'remove individual key/item pair by key
    dict.Remove dict.Keys(0)                    'remove first key/item by index position
    dict.Remove dict.Keys(UBound(dict.Keys))    'remove last key/item by index position
    dict.RemoveAll                              'remove all remaining key/item pairs

End Sub
```

Агрегация данных с помощью Scripting.Dictionary (Maximum, Count)

Словари отлично подходят для управления информацией, в которой происходит несколько записей, но для каждого набора записей используется только одно значение - первое или последнее значение, минимальное или максимальное значение, среднее значение, сумма и т. Д.

Рассмотрим книгу, в которой хранится журнал активности пользователя, со сценарием, который вставляет имя пользователя и дату редактирования каждый раз, когда кто-то

редактирует книгу:

`Log` лист

	B
боб	10/12/2016 9:00
Алиса	13.10.2013 13:00
боб	13.10.2006 13:30
Алиса	13.10.2016 14:00
Алиса	14.10.2013 13:00

Предположим, вы хотите вывести последнее время редактирования для каждого пользователя в рабочий лист с именем `Summary` .

Заметки:

1. Предполагается, что данные находятся в `ActiveWorkbook` .
2. Мы используем массив для вытягивания значений из листа; это более эффективно, чем повторение каждой ячейки.
3. `Dictionary` создается с использованием раннего связывания.

```
Sub LastEdit()
Dim vLog as Variant, vKey as Variant
Dim dict as New Scripting.Dictionary
Dim lastRow As Integer, lastColumn As Integer
Dim i as Long
Dim anchor As Range

With ActiveWorkbook
    With .Sheets("Log")
        'Pull entries in "log" into a variant array
        lastRow = .Range("a" & .Rows.Count).End(xlUp).Row
        vlog = .Range("a1", .Cells(lastRow, 2)).Value2

        'Loop through array
        For i = 1 to lastRow
            Dim username As String
            username = vlog(i, 1)
            Dim editDate As Date
            editDate = vlog(i, 2)

            'If the username is not yet in the dictionary:
            If Not dict.Exists(username) Then
                dict(username) = editDate
            ElseIf dict(username) < editDate Then
                dict(username) = editDate
            End If
        Next
    End With
```

```
    With .Sheets("Summary")
        'Loop through keys
        For Each vKey in dict.Keys
            'Add the key and value at the next available row
            Anchor = .Range("A" & .Rows.Count).End(xlUp).Offset(1,0)
            Anchor = vKey
            Anchor.Offset(0,1) = dict(vKey)
        Next vKey
    End With
End With
End Sub
```

и выход будет выглядеть так:

`Summary` рабочий лист

	B
боб	13.10.2006 13:30
Алиса	14.10.2013 13:00

Если, с другой стороны, вы хотите вывести, сколько раз каждый пользователь редактировал книгу, тело цикла `For` должно выглядеть так:

```
        'Loop through array
        For i = 1 to lastRow
            Dim username As String
            username = vlog(i, 1)

            'If the username is not yet in the dictionary:
            If Not dict.Exists(username) Then
                dict(username) = 1
            Else
                dict(username) = dict(username) + 1
            End If
        Next
```

и выход будет выглядеть так:

`Summary` рабочий лист

	B
боб	2
Алиса	3

Получение уникальных значений с помощью Scripting.Dictionary

`Dictionary` позволяет очень просто получить уникальный набор значений. Рассмотрим следующую функцию:

```
Function Unique(values As Variant) As Variant()
    'Put all the values as keys into a dictionary
    Dim dict As New Scripting.Dictionary
    Dim val As Variant
    For Each val In values
        dict(val) = 1 'The value doesn't matter here
    Next
    Unique = dict.Keys
End Function
```

который вы могли бы затем вызвать следующим образом:

```
Dim duplicates() As Variant
duplicates = Array(1, 2, 3, 1, 2, 3)
Dim uniqueVals() As Variant
uniqueVals = Unique(duplicates)
```

и `uniqueVals` будет содержать только `{1,2,3}` .

Примечание. Эта функция может использоваться с любым перечислимым объектом.

Прочитайте Объект Scripting.Dictionary онлайн: https://riptutorial.com/ru/vba/topic/3667/объект-scripting-dictionary

глава 21: Объектно-ориентированный VBA

Examples

абстракция

Уровни абстракции помогают определить, когда разделить вещи.

Абстракция достигается за счет внедрения функциональности со все более подробным кодом. Точкой входа макроса должна быть небольшая процедура с *высоким уровнем абстракции,* которая позволяет легко понять, что происходит:

```
Public Sub DoSomething()
    With New SomeForm
        Set .Model = CreateViewModel
        .Show vbModal
        If .IsCancelled Then Exit Sub
        ProcessUserData .Model
    End With
End Sub
```

Процедура `DoSomething` имеет высокий *уровень абстракции* : мы можем сказать, что она отображает форму и создает некоторую модель и передает этот объект некоторой процедуре `ProcessUserData` , которая знает, что с ней делать - то, как создается модель, - это работа другой процедуры:

```
Private Function CreateViewModel() As ISomeModel
    Dim result As ISomeModel
    Set result = SomeModel.Create(Now, Environ$("UserName"))
    result.AvailableItems = GetAvailableItems
    Set CreateViewModel = result
End Function
```

Функция `CreateViewModel` отвечает только за создание экземпляра `ISomeModel` . Часть этой ответственности состоит в том, чтобы получить массив *доступных элементов* - как эти предметы были приобретены - это деталь реализации, которая абстрагируется после процедуры `GetAvailableItems` :

```
Private Function GetAvailableItems() As Variant
    GetAvailableItems = DataSheet.Names("AvailableItems").RefersToRange
End Function
```

Здесь процедура считывает доступные значения из именованного диапазона на листе `DataSheet` . Это также можно было бы прочитать из базы данных, или значения могут быть жестко закодированы: это *деталь реализации,* которая не вызывает беспокойства ни для одного из более высоких уровней абстракции.

Инкапсуляция

Инкапсуляция скрывает детали реализации из кода клиента.

Пример обработки QueryClose демонстрирует инкапсуляцию: форма имеет элемент управления флажком , но его клиентский код не работает с ним напрямую - флажок - это *деталь реализации* , что должен знать код клиента, является ли параметр включен или нет.

Когда значение флажка изменяется, обработчик назначает частный член поля:

```
Private Type TView
    IsCancelled As Boolean
    SomeOtherSetting As Boolean
    'other properties skipped for brievety
End Type
Private this As TView

'...

Private Sub SomeOtherSettingInput_Change()
    this.SomeOtherSetting = CBool(SomeOtherSettingInput.Value)
End Sub
```

И когда клиентский код хочет прочитать это значение, ему не нужно беспокоиться о флажке - вместо этого он просто использует свойство `SomeOtherSetting` :

```
Public Property Get SomeOtherSetting() As Boolean
    SomeOtherSetting = this.SomeOtherSetting
End Property
```

Свойство `SomeOtherSetting` *инкапсулирует* состояние флажка; клиентский код не обязательно должен знать, что есть флажок, только что существует параметр с логическим значением. *Инкапсулируя* `Boolean` значение, мы добавили *слой абстракции* вокруг флажка.

Использование интерфейсов для обеспечения неизменности

Давайте толкать , что шаг *инкапсулируя модель* формы в специальном модуле класса. Но если мы создали `Public Property` для `UserName` и `Timestamp` , нам нужно было бы открыть `Property Let` accessors, изменив свойства, и мы не хотим, чтобы клиентский код имел возможность изменять эти значения после их установки.

Функция `CreateViewModel` в примере **абстракции** возвращает класс `ISomeModel` : это наш *интерфейс* , и он выглядит примерно так:

```
Option Explicit
```

```
Public Property Get Timestamp() As Date
End Property

Public Property Get UserName() As String
End Property

Public Property Get AvailableItems() As Variant
End Property

Public Property Let AvailableItems(ByRef value As Variant)
End Property

Public Property Get SomeSetting() As String
End Property

Public Property Let SomeSetting(ByVal value As String)
End Property

Public Property Get SomeOtherSetting() As Boolean
End Property

Public Property Let SomeOtherSetting(ByVal value As Boolean)
End Property
```

Значения `Timestamp` и `UserName` отображают только атрибут `Property Get` . Теперь класс `SomeModel` может реализовать этот интерфейс:

```
Option Explicit
Implements ISomeModel

Private Type TModel
    Timestamp As Date
    UserName As String
    SomeSetting As String
    SomeOtherSetting As Boolean
    AvailableItems As Variant
End Type
Private this As TModel

Private Property Get ISomeModel_Timestamp() As Date
    ISomeModel_Timestamp = this.Timestamp
End Property

Private Property Get ISomeModel_UserName() As String
    ISomeModel_UserName = this.UserName
End Property

Private Property Get ISomeModel_AvailableItems() As Variant
    ISomeModel_AvailableItems = this.AvailableItems
End Property

Private Property Let ISomeModel_AvailableItems(ByRef value As Variant)
    this.AvailableItems = value
End Property

Private Property Get ISomeModel_SomeSetting() As String
    ISomeModel_SomeSetting = this.SomeSetting
End Property
```

```
Private Property Let ISomeModel_SomeSetting(ByVal value As String)
    this.SomeSetting = value
End Property

Private Property Get ISomeModel_SomeOtherSetting() As Boolean
    ISomeModel_SomeOtherSetting = this.SomeOtherSetting
End Property

Private Property Let ISomeModel_SomeOtherSetting(ByVal value As Boolean)
    this.SomeOtherSetting = value
End Property

Public Property Get Timestamp() As Date
    Timestamp = this.Timestamp
End Property

Public Property Let Timestamp(ByVal value As Date)
    this.Timestamp = value
End Property

Public Property Get UserName() As String
    UserName = this.UserName
End Property

Public Property Let UserName(ByVal value As String)
    this.UserName = value
End Property

Public Property Get AvailableItems() As Variant
    AvailableItems = this.AvailableItems
End Property

Public Property Let AvailableItems(ByRef value As Variant)
    this.AvailableItems = value
End Property

Public Property Get SomeSetting() As String
    SomeSetting = this.SomeSetting
End Property

Public Property Let SomeSetting(ByVal value As String)
    this.SomeSetting = value
End Property

Public Property Get SomeOtherSetting() As Boolean
    SomeOtherSetting = this.SomeOtherSetting
End Property

Public Property Let SomeOtherSetting(ByVal value As Boolean)
    this.SomeOtherSetting = value
End Property
```

Все члены интерфейса являются `Private` , и все члены интерфейса должны быть реализованы для компиляции кода. `Public` члены не являются частью интерфейса и поэтому не подвергаются действию кода, написанного против интерфейса `ISomeModel` .

Использование фабричного метода для моделирования конструктора

Используя атрибут VB_PredeclaredId , мы можем сделать класс `SomeModel` *экземпляром по умолчанию* и написать функцию, которая работает как член уровня (`Shared` in VB.NET, `static` in C #), который клиентский код может вызывать без необходимости сначала создавать экземпляр, как мы здесь делали:

```
Private Function CreateViewModel() As ISomeModel
    Dim result As ISomeModel
    Set result = SomeModel.Create(Now, Environ$("UserName"))
    result.AvailableItems = GetAvailableItems
    Set CreateViewModel = result
End Function
```

Этот *заводский метод* присваивает значения свойств, доступные только для чтения, при доступе от интерфейса `ISomeModel` , здесь `Timestamp` и `UserName` :

```
Public Function Create(ByVal pTimeStamp As Date, ByVal pUserName As String) As ISomeModel
    With New SomeModel
        .Timestamp = pTimeStamp
        .UserName = pUserName
        Set Create = .Self
    End With
End Function

Public Property Get Self() As ISomeModel
    Set Self = Me
End Property
```

И теперь мы можем `ISomeModel` интерфейс `ISomeModel` , который предоставляет `Timestamp` и `UserName` как свойства только для чтения, которые никогда не могут быть переназначены (пока код написан против интерфейса).

Полиморфизм

Полиморфизм - это способность представить один и тот же интерфейс для разных базовых реализаций.

Возможность реализации интерфейсов позволяет полностью развязать логику приложения из пользовательского интерфейса или из базы данных или из того или иного рабочего листа.

Скажем, у вас есть интерфейс `ISomeView` который реализует сама форма:

```
Option Explicit

Public Property Get IsCancelled() As Boolean
End Property

Public Property Get Model() As ISomeModel
End Property

Public Property Set Model(ByVal value As ISomeModel)
```

```
End Property

Public Sub Show()
End Sub
```

Кодировка кода формы может выглядеть так:

```
Option Explicit
Implements ISomeView

Private Type TView
    IsCancelled As Boolean
    Model As ISomeModel
End Type
Private this As TView

Private Property Get ISomeView_IsCancelled() As Boolean
    ISomeView_IsCancelled = this.IsCancelled
End Property

Private Property Get ISomeView_Model() As ISomeModel
    Set ISomeView_Model = this.Model
End Property

Private Property Set ISomeView_Model(ByVal value As ISomeModel)
    Set this.Model = value
End Property

Private Sub ISomeView_Show()
    Me.Show vbModal
End Sub

Private Sub SomeOtherSettingInput_Change()
    this.Model.SomeOtherSetting = CBool(SomeOtherSettingInput.Value)
End Sub

'...other event handlers...

Private Sub OkButton_Click()
    Me.Hide
End Sub

Private Sub CancelButton_Click()
    this.IsCancelled = True
    Me.Hide
End Sub

Private Sub UserForm_QueryClose(Cancel As Integer, CloseMode As Integer)
    If CloseMode = VbQueryClose.vbFormControlMenu Then
        Cancel = True
        this.IsCancelled = True
        Me.Hide
    End If
End Sub
```

Но тогда ничто не запрещает создание другого модуля класса, который реализует интерфейс `ISomeView` *без пользовательской формы* - это может быть класс `SomeViewMock` :

```
Option Explicit
Implements ISomeView

Private Type TView
    IsCancelled As Boolean
    Model As ISomeModel
End Type
Private this As TView

Public Property Get IsCancelled() As Boolean
    IsCancelled = this.IsCancelled
End Property

Public Property Let IsCancelled(ByVal value As Boolean)
    this.IsCancelled = value
End Property

Private Property Get ISomeView_IsCancelled() As Boolean
    ISomeView_IsCancelled = this.IsCancelled
End Property

Private Property Get ISomeView_Model() As ISomeModel
    Set ISomeView_Model = this.Model
End Property

Private Property Set ISomeView_Model(ByVal value As ISomeModel)
    Set this.Model = value
End Property

Private Sub ISomeView_Show()
    'do nothing
End Sub
```

И теперь мы можем изменить код, который работает с `UserForm` и заставить его работать с интерфейсом `ISomeView` , например, предоставив ему форму как параметр, а не создавая ее:

```
Public Sub DoSomething(ByVal view As ISomeView)
    With view
        Set .Model = CreateViewModel
        .Show
        If .IsCancelled Then Exit Sub
        ProcessUserData .Model
    End With
End Sub
```

Поскольку метод `DoSomething` зависит от интерфейса (то есть *абстракции*), а не от *конкретного класса* (например, определенного `UserForm`), мы можем написать автоматизированный модульный тест, который гарантирует, что `ProcessUserData` не будет выполняться, когда `view.IsCancelled` is `True` , сделав нашу test создайте экземпляр `SomeViewMock` , установив свойство `IsCancelled` в значение `True` и передав его в `DoSomething` .

Тестовый код зависит от абстракций

Пробные тесты в VBA можно сделать, есть надстройки, которые даже интегрируют его в

среду IDE. Но когда код *тесно связан* с рабочим листом, базой данных, формой или файловой системой, тогда в модульном тесте начинает требоваться фактический рабочий лист, база данных, форма или файловая система - и эти *зависимости* являются новым отказом вне контроля указывает, что тестируемый код должен изолироваться, так что модульные тесты *не* требуют фактического рабочего листа, базы данных, формы или файловой системы.

Путем написания кода с интерфейсов таким образом, чтобы тестовый код мог *вводить реализацию* заглушки / макета (например, вышеприведенный пример `SomeViewMock`), вы можете писать тесты в «контролируемой среде» и имитировать, что происходит, когда каждый из 42 возможных перестановки пользовательских взаимодействий на данные формы, даже не отображая форму и вручную щелкая на элементе управления формой.

Прочитайте Объектно-ориентированный VBA онлайн: https://riptutorial.com/ru/vba/topic/5357/объектно-ориентированный-vba

глава 22: Объявление и назначение строк

замечания

Строки являются ссылочным типом и являются центральными для большинства задач программирования. Строкам присваивается текст, даже если текст является числовым. Строки могут быть нулевой длины или любой длины до 2 ГБ. Современные версии VBA хранят строки внутри с использованием байтового массива байтов байтов байтов (альтернатива Unicode).

Examples

Объявить строчную константу

```
Const appName As String = "The App For That"
```

Объявление переменной переменной ширины

```
Dim surname As String 'surname can accept strings of variable length
surname = "Smith"
surname = "Johnson"
```

Объявить и назначить строку с фиксированной шириной

```
'Declare and assign a 1-character fixed-width string
Dim middleInitial As String * 1 'middleInitial must be 1 character in length
middleInitial = "M"

'Declare and assign a 2-character fixed-width string `stateCode`,
'must be 2 characters in length
Dim stateCode As String * 2
stateCode = "TX"
```

Объявить и присвоить массив строк

```
'Declare, dimension and assign a string array with 3 elements
Dim departments(2) As String
departments(0) = "Engineering"
departments(1) = "Finance"
departments(2) = "Marketing"

'Declare an undimensioned string array and then dynamically assign with
'the results of a function that returns a string array
Dim stateNames() As String
stateNames = VBA.Strings.Split("Texas;California;New York", ";")
```

```
'Declare, dimension and assign a fixed-width string array
Dim stateCodes(2) As String * 2
stateCodes(0) = "TX"
stateCodes(1) = "CA"
stateCodes(2) = "NY"
```

Назначение определенных символов в строке с помощью оператора Mid

VBA предлагает функцию Mid для *возврата* подстрок внутри строки, но также предлагает Mid *Statement,* который может использоваться для назначения подстрок или отдельных символов с строкой.

Функция `Mid` обычно появляется в правой части оператора присваивания или в состоянии, но `Mid` Statement обычно появляется в левой части оператора присваивания.

```
Dim surname As String
surname = "Smith"

'Use the Mid statement to change the 3rd character in a string
Mid(surname, 3, 1) = "y"
Debug.Print surname

'Output:
'Smyth
```

Примечание. Если вам нужно назначить отдельные *байты* в строке вместо отдельных *символов* внутри строки (см. Примечания ниже относительно `MidB` набора символов), можно использовать инструкцию `MidB` . В этом случае второй аргумент для оператора `MidB` - это позиция байта, основанная на 1, где будет `MidB(surname, 5, 2) = "y"` замена, так что эквивалентная строка в примере выше будет `MidB(surname, 5, 2) = "y"` .

Назначение в массив байтов и из него

Строки могут быть назначены непосредственно байт-массивам и наоборот. Помните, что строки хранятся в многобайтовом наборе символов (см. Примечания ниже), поэтому только каждый другой индекс результирующего массива будет частью символа, который попадает в диапазон ASCII.

```
Dim bytes() As Byte
Dim example As String

example = "Testing."
bytes = example             'Direct assignment.

'Loop through the characters. Step 2 is used due to wide encoding.
Dim i As Long
For i = LBound(bytes) To UBound(bytes) Step 2
    Debug.Print Chr$(bytes(i))  'Prints T, e, s, t, i, n, g, .
Next

Dim reverted As String
```

```
reverted = bytes            'Direct assignment.
Debug.Print reverted        'Prints "Testing."
```

Прочитайте Объявление и назначение строк онлайн: https://riptutorial.com/ru/vba/topic/3446/объявление-и-назначение-строк

глава 23: Объявление переменных

Examples

Неявная и явная декларация

Если модуль кода не содержит `Option Explicit` в верхней части модуля, тогда компилятор автоматически (то есть «неявно») создает переменные для вас, когда вы их используете. Они будут использовать переменную типа `Variant` .

```
Public Sub ExampleDeclaration()

    someVariable = 10                   '
    someOtherVariable = "Hello World"
    'Both of these variables are of the Variant type.

End Sub
```

В приведенном выше коде, если указан параметр `Option Explicit` , код будет прерываться, потому что ему не нужны необходимые инструкции `Dim` для `someVariable` и `someOtherVariable` .

```
Option Explicit

Public Sub ExampleDeclaration()

    Dim someVariable As Long
    someVariable = 10

    Dim someOtherVariable As String
    someOtherVariable = "Hello World"

End Sub
```

Рекомендуется использовать Option Explicit в модулях кода, чтобы гарантировать, что вы объявите все переменные.

См. VBA Best Practices, как установить этот параметр по умолчанию.

переменные

Объем

Переменная может быть объявлена (при увеличении уровня видимости):

- На уровне процедуры, используя ключевое слово `Dim` в любой процедуре; *локальная переменная* .
- На уровне модуля, используя ключевое слово `Private` в модуле любого типа; *частное*

поле .

- На уровне экземпляра используйте ключевое слово `Friend` в модуле любого типа; *поле друга* .
- На уровне экземпляра, используя ключевое слово `Public` в модуле класса любого типа; *публичное поле* .
- Глобально, используя ключевое слово `Public` в *стандартном модуле* ; *глобальная переменная* .

Переменные всегда должны быть объявлены с наименьшей возможной областью: предпочитайте передавать параметры процедурам, а не объявлять глобальные переменные.

Дополнительную информацию см. В разделе Модификаторы доступа .

Местные переменные

Используйте ключевое слово `Dim` чтобы объявить *локальную переменную* :

```
Dim identifierName [As Type][, identifierName [As Type], ...]
```

Параметр `[As Type]` синтаксиса объявления является необязательным. Когда указано, он устанавливает тип данных переменной, который определяет, сколько памяти будет выделено этой переменной. Это объявляет переменную `String` :

```
Dim identifierName As String
```

Если тип не указан, тип неявно `Variant` :

```
Dim identifierName 'As Variant is implicit
```

Синтаксис VBA также поддерживает объявление нескольких переменных в одном выражении:

```
Dim someString As String, someVariant, someValue As Long
```

Обратите внимание, что `[As Type]` должен быть указан для каждой переменной (кроме « Variant»). Это относительно распространенная ловушка:

```
Dim integer1, integer2, integer3 As Integer 'Only integer3 is an Integer.
                                            'The rest are Variant.
```

Статические переменные

Локальные переменные также могут быть `Static` . В VBA ключевое слово `Static`

используется для того, чтобы переменная «запомнила» значение, которое она имела, в последний раз, когда была вызвана процедура:

```
Private Sub DoSomething()
    Static values As Collection
    If values Is Nothing Then
        Set values = New Collection
        values.Add "foo"
        values.Add "bar"
    End If
    DoSomethingElse values
End Sub
```

Здесь коллекция `values` объявляется как `Static` локальная; потому что это *переменная объекта* , она инициализируется `Nothing` . Условие, следующее за объявлением, проверяет, была ли ссылка на объект `Set` раньше - если она выполняется в первый раз, процедура инициализируется. `DoSomethingElse` может добавлять или удалять элементы, и они все равно будут в коллекции в следующий раз, когда вызывается `DoSomething` .

альтернатива

> Ключевое слово VBA `Static` легко может быть неправильно понято - *особенно* опытные программисты, которые обычно работают на других языках. На многих языках `static` используется для того, чтобы член класса (field, property, method, ...) принадлежал *типу,* а не *экземпляру* . Код в `static` контексте не может ссылаться на код в контексте *экземпляра* . Ключевое слово VBA `Static` означает что-то совершенно другое.

Часто `Static` локальный объект может быть также реализован как `Private` переменная (поле) на уровне модуля, однако это бросает вызов принципу, с помощью которого переменная должна быть объявлена с наименьшей возможной областью; доверяйте своим инстинктам, используйте то, что вы предпочитаете - оба будут работать ... но использование `Static` без понимания того, что он делает, может привести к интересным ошибкам.

Дим против частного

Ключевое слово `Dim` является законным на уровне процедур и модулей; его использование на уровне модуля эквивалентно использованию ключевого слова `Private` :

```
Option Explicit
Dim privateField1 As Long 'same as Private privateField2 as Long
Private privateField2 As Long 'same as Dim privateField2 as Long
```

Ключевое слово `Private` доступно только на уровне модуля; это вызывает резервирование `Dim` для локальных переменных и объявление переменных модуля с помощью `Private` , особенно с контрастным ключевым словом `Public` которое должно быть использовано для

объявления публичного участника. Также можно использовать `Dim` *везде* - то , что имеет значение *консистенция:*

«Частные поля»

- **DO** использовать `Private` чтобы объявить переменную уровня модуля.
- **DO** используйте `Dim` чтобы объявить локальную переменную.
- **НЕ** используйте `Dim` чтобы объявить переменную уровня модуля.

«Тусклый везде»

- **DO** используйте `Dim` чтобы объявить что-либо частным / местным.
- **НЕ** используйте `Private` чтобы объявить переменную уровня модуля.
- **ИЗБЕГАЙТЕ** объявление `Public` полей. *

* В общем, следует избегать объявления `Public` или `Global` полей.

поля

Переменная, объявленная на уровне модуля, в разделе *объявлений* в верхней части тела модуля является *полем* . `Public` поле, объявленное в *стандартном модуле,* является *глобальной переменной* :

```
Public PublicField As Long
```

Доступ к переменной с глобальной областью можно получить из любого места, включая другие проекты VBA, которые будут ссылаться на проект, в котором он объявлен.

Чтобы сделать переменную global / public, но только видимую изнутри проекта, используйте модификатор `Friend` :

```
Friend FriendField As Long
```

Это особенно полезно в надстройках, где намерение заключается в том, что другие проекты VBA ссылаются на проект надстройки и могут потреблять открытый API.

```
Friend FriendField As Long 'public within the project, aka for "friend" code
Public PublicField As Long 'public within and beyond the project
```

Поля друзей недоступны в стандартных модулях.

Поля экземпляра

Переменная, объявленная на уровне модуля, в разделе *объявлений* в верхней части тела

модуля класса (включая `ThisWorkbook` , `ThisDocument` , `Worksheet` , `UserForm` и *модулях классов*) является *полем экземпляра* : оно существует только до тех пор, пока существует *экземпляр* класс вокруг.

```
'> Class1
Option Explicit
Public PublicField As Long
```

```
'> Module1
Option Explicit
Public Sub DoSomething()
    'Class1.PublicField means nothing here
    With New Class1
        .PublicField = 42
    End With
    'Class1.PublicField means nothing here
End Sub
```

Инкапсулирующие поля

Данные Instance часто хранится `Private` и дублирован *инкапсулируются.* Закрытое поле можно открыть с помощью процедуры `Property` . Чтобы публично публиковать приватную переменную, не предоставляя доступ к записи вызывающему, модуль класса (или стандартный модуль) реализует элемент « `Property Get` :

```
Option Explicit
Private encapsulated As Long

Public Property Get SomeValue() As Long
    SomeValue = encapsulated
End Property

Public Sub DoSomething()
    encapsulated = 42
End Sub
```

Сам класс может изменить инкапсулированное значение, но вызывающий код может получить доступ только к `Public` членам (и членам `Friend` , если вызывающий объект находится в одном проекте).

Чтобы разрешить вызывающему абоненту:

- Инкапсулированное **значение** , модуль предоставляет элемент `Property Let` .
- **Ссылка на** инкапсулированные **объекты** , модуль предоставляет член `Property Set` .

Константы (Const)

Если у вас есть значение, которое никогда не изменяется в вашем приложении, вы можете определить именованную константу и использовать ее вместо буквенного значения.

Вы можете использовать Const только на уровне модуля или процедуры. Это означает, что контекст объявления для переменной должен быть классом, структурой, модулем, процедурой или блоком и не может быть исходным файлом, пространством имен или интерфейсом.

```
Public Const GLOBAL_CONSTANT As String = "Project Version #1.000.000.001"
Private Const MODULE_CONSTANT As String = "Something relevant to this Module"

Public Sub ExampleDeclaration()

    Const SOME_CONSTANT As String = "Hello World"

    Const PI As Double = 3.141592653

End Sub
```

Хотя можно считать хорошей практикой указывать константные типы, это строго не требуется. Не указывая тип, он все равно приведет к правильному типу:

```
Public Const GLOBAL_CONSTANT = "Project Version #1.000.000.001" 'Still a string
Public Sub ExampleDeclaration()

    Const SOME_CONSTANT = "Hello World"           'Still a string
    Const DERIVED_CONSTANT = SOME_CONSTANT        'DERIVED_CONSTANT is also a string
    Const VAR_CONSTANT As Variant = SOME_CONSTANT 'VAR_CONSTANT is Variant/String

    Const PI = 3.141592653        'Still a double
    Const DERIVED_PI = PI         'DERIVED_PI is also a double
    Const VAR_PI As Variant = PI  'VAR_PI is Variant/Double

End Sub
```

Обратите внимание, что это специфично для констант и в отличие от переменных, где не указано, что тип приводит к типу Variant.

Хотя можно явно объявить константу как String, невозможно объявить константу в виде строки с использованием синтаксиса строки фиксированной ширины

```
'This is a valid 5 character string constant
Const FOO As String = "ABCDE"

'This is not valid syntax for a 5 character string constant
Const FOO As String * 5 = "ABCDE"
```

Модификаторы доступа

`Dim` должна быть зарезервирована для локальных переменных. На уровне модуля предпочитайте явные модификаторы доступа:

- `Private` для частных полей, доступ к которым возможен только в том модуле, в котором они объявлены.
- `Public`

для общедоступных полей и глобальных переменных, к которым может обращаться любой вызывающий код.
- `Friend` для переменных, публикуемых в рамках проекта, но недоступных для других проектов VBA, связанных с надстройками (для надстроек)
- `Global` также может использоваться для `Public` полей в стандартных модулях, но является незаконным в модулях классов и в любом случае является устаревшим - вместо этого предпочитайте `Public` модификатор. Этот модификатор также не является законным для процедур.

Модификаторы доступа применимы как к переменным, так и к процедурам.

```
Private ModuleVariable As String
Public GlobalVariable As String

Private Sub ModuleProcedure()

    ModuleVariable = "This can only be done from within the same Module"

End Sub

Public Sub GlobalProcedure()

    GlobalVariable = "This can be done from any Module within this Project"

End Sub
```

Дополнительный модуль

Открытые `Sub` процедуры в стандартных модулях отображаются как макросы и могут быть прикреплены к элементам управления и сочетания клавиш в документе хоста.

И наоборот, публичные процедуры `Function` в стандартных модулях отображаются как пользовательские функции (UDF) в хост-приложении.

Указание отдельного модуля `Option Private Module` в верхней части стандартного модуля не позволяет его членам отображаться как макросы и UDF для приложения-хозяина.

Тип подсказки

Тип подсказки **сильно** обескуражен. Они существуют и документируются здесь по причинам исторической и обратной совместимости. `As [DataType]` этого следует использовать синтаксис `As [DataType]` .

```
Public Sub ExampleDeclaration()

        Dim someInteger% '% Equivalent to "As Integer"
        Dim someLong&    '& Equivalent to "As Long"
        Dim someDecimal@ '@ Equivalent to "As Currency"
```

```
        Dim someSingle!  '! Equivalent to "As Single"
        Dim someDouble#  '# Equivalent to "As Double"
        Dim someString$  '$ Equivalent to "As String"

        Dim someLongLong^  '^ Equivalent to "As LongLong" in 64-bit VBA hosts
End Sub
```

Типовые подсказки значительно уменьшают читаемость кода и поощряют венгерскую нотацию, которая *также* препятствует читаемости:

```
Dim strFile$
Dim iFile%
```

Вместо этого, объявляйте переменные ближе к их использованию и называйте вещи тем, что они используются, а не после их типа:

```
Dim path As String
Dim handle As Integer
```

Типовые подсказки также могут использоваться в литералах, чтобы обеспечить соблюдение определенного типа. По умолчанию числовой литерал, меньший, чем 32 768, будет интерпретироваться как литерал `Integer` , но с подсказкой типа вы можете контролировать это:

```
Dim foo 'implicit Variant
foo = 42& ' foo is now a Long
foo = 42# ' foo is now a Double
Debug.Print TypeName(42!) ' prints "Single"
```

Типные подсказки обычно не нужны для литералов, потому что они будут назначены переменной, объявленной с явным типом, или неявно преобразуются в соответствующий тип при передаче в качестве параметров. Неявные преобразования можно избежать, используя одну из явных функций преобразования типов:

```
'Calls procedure DoSomething and passes a literal 42 as a Long using a type hint
DoSomething 42&

'Calls procedure DoSomething and passes a literal 42 explicitly converted to a Long
DoSomething CLng(42)
```

Строковые возвращаемые встроенные функции

Большинство встроенных функций, которые обрабатывают строки, представлены в двух версиях: свободно напечатанная версия, которая возвращает `Variant` , и строго

типизированную версию (заканчивающуюся на `$`), которая возвращает `String` . Если вы не назначаете возвращаемое значение для `Variant` , вы должны предпочесть версию, которая возвращает `String` противном случае происходит неявное преобразование возвращаемого значения.

```
Debug.Print Left(foo, 2)  'Left returns a Variant
Debug.Print Left$(foo, 2) 'Left$ returns a String
```

Эти функции:

- VBA.Conversion.Error -> VBA.Conversion.Error $
- VBA.Conversion.Hex -> VBA.Conversion.Hex $
- VBA.Conversion.Oct -> VBA.Conversion.Oct $
- VBA.Conversion.Str -> VBA.Conversion.Str $
- VBA.FileSystem.CurDir -> VBA.FileSystem.CurDir $
- VBA. [_ HiddenModule] .Input -> VBA. [_ HiddenModule] .Input $
- VBA. [_ HiddenModule] .InputB -> VBA. [_ HiddenModule] .InputB $
- VBA.Interaction.Command -> VBA.Interaction.Command $
- VBA.Interaction.Environ -> VBA.Interaction.Environ $
- VBA.Strings.Chr -> VBA.Strings.Chr $
- VBA.Strings.ChrB -> VBA.Strings.ChrB $
- VBA.Strings.ChrW -> VBA.Strings.ChrW $
- VBA.Strings.Format -> VBA.Strings.Format $
- VBA.Strings.LCase -> VBA.Strings.LCase $
- VBA.Strings.Left -> VBA.Strings.Left $
- VBA.Strings.LeftB -> VBA.Strings.LeftB $
- VBA.Strings.LTtrim -> VBA.Strings.LTrim $
- VBA.Strings.Mid -> VBA.Strings.Mid $
- VBA.Strings.MidB -> VBA.Strings.MidB $
- VBA.Strings.Right -> VBA.Strings.Right $
- VBA.Strings.RightB -> VBA.Strings.RightB $
- VBA.Strings.RTrim -> VBA.Strings.RTrim $
- VBA.Strings.Space -> VBA.Strings.Space $
- VBA.Strings.Str -> VBA.Strings.Str $
- VBA.Strings.String -> VBA.Strings.String $
- VBA.Strings.Trim -> VBA.Strings.Trim $
- VBA.Strings.UCase -> VBA.Strings.UCase $

Обратите внимание, что это *псевдонимы* функций, не совсем *намеки на тип* . Функция `Left` соответствует скрытой функции `B_Var_Left` , а версия `Left$` соответствует скрытой функции `B_Str_Left` .

В ранних версиях VBA знак `$` не является допустимым символом, а имя функции должно быть заключено в квадратные скобки. В Word Basic было много и много функций, которые возвращали строки, которые заканчивались в $.

Объявление строк фиксированной длины

В VBA строки могут быть объявлены с определенной длиной; они автоматически дополняются или усекаются, чтобы поддерживать указанную длину.

```
Public Sub TwoTypesOfStrings()

    Dim FixedLengthString As String * 5 ' declares a string of 5 characters
    Dim NormalString As String

    Debug.Print FixedLengthString       ' Prints "     "
    Debug.Print NormalString            ' Prints ""

    FixedLengthString = "123"           ' FixedLengthString now equals "123  "
    NormalString = "456"                ' NormalString now equals "456"

    FixedLengthString = "123456"        ' FixedLengthString now equals "12345"
    NormalString = "456789"             ' NormalString now equals "456789"

End Sub
```

Когда использовать статическую переменную

Статическая переменная, объявленная локально, не разрушается и не теряет своего значения при выходе из процедуры Sub. Последующие вызовы процедуры не требуют повторной инициализации или присвоения, хотя вы можете «нулевать» любые запоминаемые значения.

Они особенно полезны при позднем связывании объекта в вспомогательном подэлементе, который вызывается повторно.

Фрагмент 1: повторное использование объекта Scripting.Dictionary на многих листах

```
Option Explicit

Sub main()
    Dim w As Long

    For w = 1 To Worksheets.Count
        processDictionary ws:=Worksheets(w)
    Next w
End Sub

Sub processDictionary(ws As Worksheet)
    Dim i As Long, rng As Range
    Static dict As Object

    If dict Is Nothing Then
        'initialize and set the dictionary object
        Set dict = CreateObject("Scripting.Dictionary")
        dict.CompareMode = vbTextCompare
    Else
        'remove all pre-existing dictionary entries
        ' this may or may not be desired if a single dictionary of entries
```

```
        ' from all worksheets is preferred
        dict.RemoveAll
    End If

    With ws

        'work with a fresh dictionary object for each worksheet
        ' without constructing/destructing a new object each time
        ' or do not clear the dictionary upon subsequent uses and
        ' build a dictionary containing entries from all worksheets

    End With
End Sub
```

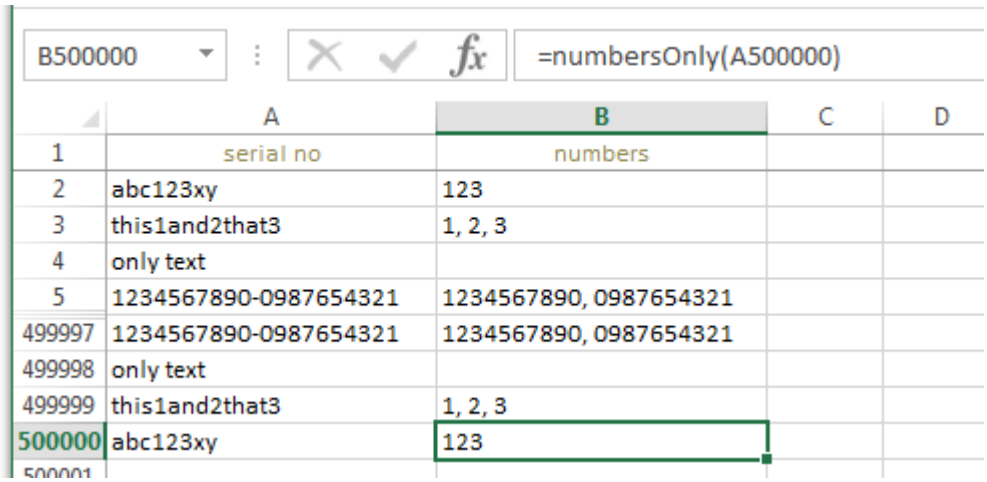
Snippet 2: создать рабочий лист UDF, который опоздает с привязкой к объекту VBScript.RegExp

```
Option Explicit

Function numbersOnly(str As String, _
                     Optional delim As String = ", ")
    Dim n As Long, nums() As Variant
    Static rgx As Object, cmat As Object

    'with rgx as static, it only has to be created once
    'this is beneficial when filling a long column with this UDF
    If rgx Is Nothing Then
        Set rgx = CreateObject("VBScript.RegExp")
    Else
        Set cmat = Nothing
    End If

    With rgx
        .Global = True
        .MultiLine = True
        .Pattern = "[0-9]{1,999}"
        If .Test(str) Then
            Set cmat = .Execute(str)
            'resize the nums array to accept the matches
            ReDim nums(cmat.Count - 1)
            'populate the nums array with the matches
            For n = LBound(nums) To UBound(nums)
                nums(n) = cmat.Item(n)
            Next n
            'convert the nums array to a delimited string
            numbersOnly = Join(nums, delim)
        Else
            numbersOnly = vbNullString
        End If
    End With
End Function
```



Пример UDF со статическим объектом, заполненным полмиллиона строк

* Истекшее время, чтобы заполнить 500K строк с помощью UDF:

- с **Dim rgx As Object** : 148.74 секунд
- с **Static rgx As Object** : 26.07 секунд

* Они должны учитываться только для относительного сравнения. Ваши собственные результаты будут варьироваться в зависимости от сложности и объем выполненных операций.

Помните, что UDF не рассчитывается один раз в жизни книги. Даже энергонезависимый UDF будет пересчитываться всякий раз, когда значения в пределах диапазона (ов), которые он ссылается, могут быть изменены. Каждое последующее событие пересчета только увеличивает преимущества статически объявленной переменной.

- Статическая переменная доступна для времени жизни модуля, а не для процедуры или функции, в которой она была объявлена и назначена.
- Статические переменные могут быть объявлены только локально.
- Статическая переменная содержит много одинаковых свойств переменной уровня частного модуля, но с более ограниченной областью.

Ссылки по теме: Static (Visual Basic)

Прочитайте Объявление переменных онлайн: https://riptutorial.com/ru/vba/topic/877/объявление-переменных

глава 24: операторы

замечания

Операторы оцениваются в следующем порядке:

- Математические операторы
- Побитовые операторы
- Операторы конкатенации
- Операторы сравнения
- Логические операторы

Операторы с совпадающим приоритетом оцениваются слева направо. Порядок по умолчанию можно переопределить, используя круглые скобки `(` и `)` для группировки выражений.

Examples

Математические операторы

Перечислено в порядке очередности:

знак	название	Описание
`^`	Возведение	Верните результат подъема левого операнда в силу правого операнда. Обратите внимание, что значение, возвращаемое экспоненциацией, *всегда* равно `Double` , независимо от того, какие типы значений делятся. Любое принуждение результата к переменному типу происходит ***после*** выполнения вычисления.
`/`	Раздел [1]	Возвращает результат деления левого операнда на правый операнд. Обратите внимание, что значение, возвращаемое делением, *всегда* равно `Double` , независимо от того, какие типы значений делятся. Любое принуждение результата к переменному типу происходит ***после*** выполнения вычисления.
`*`	Умножение [1]	Возвращает произведение двух операндов.
`\`	Целостный отдел	Возвращает целочисленный результат деления левого операнда правым операндом ***после*** округления обеих сторон с округлением .5. Любое остальное подразделение игнорируется. Если правый операнд (делитель) равен `0` , это приведет к ошибке

знак	название	Описание
		времени выполнения 11: деление на ноль. Обратите внимание, что это происходит ***после*** округления - выражения, такие как `3 \ 0.4` , также приводят к делению на нулевую ошибку.
`Mod`	Модульное	Возвращает целочисленный остаток деления левого операнда на правый операнд. Операнд с каждой стороны округляется до целого числа *до* деления с округлением .5. Например, и `8.6 Mod 3` и `12 Mod 2.6` приводят к `0` . Если правый операнд (делитель) равен `0` , это приведет к ошибке времени выполнения 11: деление на ноль. Обратите внимание, что это выполняется ***после*** округления - выражения, такие как `3 Mod 0.4` , также приводят к делению на нулевую ошибку.
`-`	Вычитание [2]	Возвращает результат вычитания правого операнда из левого операнда.
`+`	Добавление [2]	Возвращает сумму из двух операндов. Обратите внимание, что этот токен также рассматривается как оператор конкатенации, когда он применяется к `String` . См. **Операторы конкатенации** .

[1] Умножение и деление рассматриваются как имеющие тот же приоритет.

[2] Сложение и вычитание рассматриваются как имеющие тот же приоритет.

Операторы конкатенации

VBA поддерживает 2 разных оператора конкатенации `+` и `&` и выполняет одну и ту же функцию при использовании со `String` типами - правая `String` добавляется к концу левой `String` .

Если оператор `&` используется с переменным типом, отличным от `String` , он неявно передается в `String` перед конкатенацией.

Обратите внимание, что оператор `+` конкатенации является перегрузкой оператора `+` сложения. Поведение `+` определяется переменными типами операндов и приоритетом типов операторов. Если оба операнда напечатаны как `String` или `Variant` с `Variant String` , они объединяются:

```
Public Sub Example()
    Dim left As String
    Dim right As String

    left = "5"
    right = "5"
```

```
    Debug.Print left + right    'Prints "55"
End Sub
```

Если какая- *либо* сторона является числовым типом, а другая сторона является `String` которая может быть принудительно введена в число, приоритет типа математических операторов заставляет оператора обрабатываться как оператор сложения и добавляются числовые значения:

```
Public Sub Example()
    Dim left As Variant
    Dim right As String

    left = 5
    right = "5"

    Debug.Print left + right    'Prints 10
End Sub
```

Такое поведение может привести к тонким, трудно отлаживающим ошибкам, особенно если используются типы `Variant` , поэтому для конкатенации обычно следует использовать оператор `&` .

Операторы сравнения

знак	название	Описание
`=`	Равно	Возвращает `True` если левый и правый операнды равны. Обратите внимание, что это перегрузка оператора присваивания.
`<>`	Не равен	Возвращает `True` если левый и правый операнды не равны.
`>`	Лучше чем	Возвращает `True` если левый операнд больше правого операнда.
`<`	Меньше, чем	Возвращает `True` если левый операнд меньше правого операнда.
`>=`	Больше или равно	Возвращает `True` если если левый операнд больше или равен правому операнду.
`<=`	Меньше или равно	Возвращает `True` если левый операнд меньше или равен правому операнду.
`Is`	Справочный капитал	Возвращает значение `True` если ссылка на левый объект - это тот же экземпляр, что и ссылка на правый объект. Он также может использоваться с `Nothing` (ссылка на нулевой объект) с обеих сторон. **Примечание.** Оператор `Is` попытается принудить оба операнда к `Object` перед выполнением сравнения. Если какая-либо сторона является примитивным типом *или* `Variant` , который

знак	название	Описание
		не содержит объект (либо не-объектный подтип, либо `vtEmpty`), сравнение приведет к ошибке времени выполнения 424 - «Требуется объект». Если любой операнд принадлежит другому *интерфейсу* одного и того же объекта, сравнение вернет `True` . Если вам нужно проверить справедливость как экземпляра, так *и* интерфейса, `ObjPtr(left) = ObjPtr(right)` используйте `ObjPtr(left) = ObjPtr(right)` .

Заметки

Синтаксис VBA позволяет «цепочки» операторов сравнения, но в целом эти конструкции следует избегать. Сравнение всегда выполняется слева направо только на 2 операндах за раз, и каждое сравнение приводит к `Boolean` . Например, выражение ...

```
a = 2: b = 1: c = 0
expr = a > b > c
```

... может быть прочитан в некоторых контекстах как проверка того, является ли `b` между `a` и `c` . В VBA это оценивается следующим образом:

```
a = 2: b = 1: c = 0
expr = a > b > c
expr = (2 > 1) > 0
expr = True > 0
expr = -1 > 0 'CInt(True) = -1
expr = False
```

Любой оператор сравнения, кроме `Is` использоваться с `Object` в качестве операнда будет выполняться на возвращаемом значении `Object` «s члена по умолчанию . Если объект не имеет члена по умолчанию, сравнение приведет к ошибке времени выполнения 438 - «Объект не поддерживает его свойство или метод».

Если `Object` не инициализирован, сравнение приведет к ошибке времени выполнения 91 - «Объектная переменная или С заблокированной переменной блока».

Если литерал `Nothing` используется с любым оператором сравнения, отличным от `Is` , это приведет к ошибке компиляции - «Недопустимое использование объекта».

Если `Object` по умолчанию `Object` является *другой* *`Object`* , VBA будет постоянно вызывать элемент по умолчанию каждого последующего возвращаемого значения до тех пор, пока не будет возвращен примитивный тип или не будет поднята ошибка. Например, предположим, что у `SomeClass` есть член по умолчанию `Value` , который является экземпляром `ChildClass` с членом `ChildValue` по `ChildValue` . Сравнение...

```
Set x = New SomeClass
Debug.Print x > 42
```

... будет оцениваться как:

```
Set x = New SomeClass
Debug.Print x.Value.ChildValue > 42
```

Если либо операнд является числовым, а *другой* операндом является `String` или `Variant` подтипа `String` , будет выполнено числовое сравнение. В этом случае, если `String` не может быть отнесено к числу, результатом сравнения будет ошибка времени выполнения 13 - «Несоответствие типа».

Если ***оба*** операнда представляют собой `String` или `Variant` подтипа `String` , сравнение строк будет выполняться на основе параметра сравнения параметров модуля кода. Эти сравнения выполняются по характеру по характеру. Обратите внимание, что *символьное представление* `String` содержащей число, ***не*** совпадает с сопоставлением числовых значений:

```
Public Sub Example()
    Dim left As Variant
    Dim right As Variant

    left = "42"
    right = "5"
    Debug.Print left > right              'Prints False
    Debug.Print Val(left) > Val(right)    'Prints True
End Sub
```

По этой причине убедитесь, что переменные `String` или `Variant` передаются в числа перед выполнением численных сравнений неравенства.

Если один из операндов - это `Date` , то числовое сравнение по базовому двойному значению будет выполняться, если другой операнд является числовым или может быть преобразован в числовой тип.

Если другой операнд представляет собой `String` или `Variant` подтипа `String` который может быть перенесен в `Date` с использованием текущего языкового стандарта, `String` будет передана в `Date` . Если он не может быть применен к `Date` в текущей локали, результатом сравнения будет ошибка времени выполнения 13 - «Несоответствие типа».

Следует соблюдать осторожность при сравнении значений `Double` или `Single` и Booleans . В отличие от других числовых типов ненулевые значения нельзя считать `True` из-за поведения VBA в продвижении типа данных сравнения с использованием числа с плавающей точкой в `Double` :

```
Public Sub Example()
    Dim Test As Double

    Test = 42        Debug.Print CBool(Test)             'Prints True.
    'True is promoted to Double - Test is not cast to Boolean
    Debug.Print Test = True             'Prints False

    'With explicit casts:
    Debug.Print CBool(Test) = True      'Prints True
    Debug.Print CDbl(-1) = CDbl(True)   'Prints True
End Sub
```

Побитовые \ Логические операторы

Все логические операторы в VBA можно рассматривать как «переопределения» побитовых операторов с тем же именем. Технически они *всегда* рассматриваются как побитовые операторы. Все операторы сравнения в VBA возвращают логическое значение , которое всегда не будет содержать ни одного из его битов (`False`) или *всех* его битов (`True`). Но он будет обрабатывать значение с *любым* битом, установленным как `True` . Это означает, что результат приведения побитового результата выражения в `Boolean` (см. Операторы сравнения) всегда будет таким же, как рассматривать его как логическое выражение.

Присвоение результата выражения с использованием одного из этих операторов даст побитовый результат. Обратите внимание, что в таблицах истинности ниже `0` эквивалентно `False` и `1` эквивалентно `True` .

`And`

Возвращает `True` если выражения с обеих сторон оцениваются как `True` .

Левый Операнд	Правый операнд	Результат
0	0	0
0	1	0
1	0	0
1	1	1

`Or`

Возвращает значение `True` если любая из сторон выражения имеет значение `True` .

Левый Операнд	Правый операнд	Результат
0	0	0
0	1	1

Левый Операнд	Правый операнд	Результат
1	0	1
1	1	1

`Not`

Возвращает `True` если выражение оценивается как `False` и `False` если выражение оценивается как `True` .

Правый операнд	Результат
0	1
1	0

`Not` единственный операнд без левого операнда. Редактор Visual Basic автоматически упростит выражения с помощью аргумента левой руки. Если вы наберете ...

```
Debug.Print x Not y
```

... VBE изменит линию на:

```
Debug.Print Not x
```

Аналогичные упрощения будут сделаны для любого выражения, содержащего левый операнд (включая выражения) для `Not` .

`Xor`

Также известен как «эксклюзивный» или «эксклюзивный». Возвращает значение `True` если оба выражения оцениваются по разным результатам.

Левый Операнд	Правый операнд	Результат
0	0	0
0	1	1
1	0	1
1	1	0

Заметим, что хотя оператор `Xor` можно *использовать* как логический оператор, нет абсолютно никаких оснований для этого, поскольку он дает тот же результат, что и

оператор сравнения `<>` .

`Eqv`

Также известен как «эквивалентность». Возвращает `True` когда оба выражения оцениваются с одним и тем же результатом.

Левый Операнд	Правый операнд	Результат
0	0	1
0	1	0
1	0	0
1	1	1

Обратите внимание, что функция `Eqv` *очень* редко используется, поскольку `x Eqv y` эквивалентно гораздо более читаемому `Not (x Xor y)` .

`Imp`

Также известен как «импликация». Возвращает `True` если оба операнда одинаковы *или* второй операнд имеет значение `True` .

Левый Операнд	Правый операнд	Результат
0	0	1
0	1	1
1	0	0
1	1	1

Обратите внимание, что функция `Imp` очень редко используется. Хорошим правилом является то, что если вы не можете объяснить, что это значит, вы должны использовать другую конструкцию.

Прочитайте операторы онлайн: https://riptutorial.com/ru/vba/topic/5813/операторы

глава 25: Ошибки времени выполнения VBA

Вступление

Код, который компилирует, может по-прежнему сталкиваться с ошибками во время выполнения. В этом разделе перечислены наиболее распространенные из них, их причины и способы их устранения.

Examples

Ошибка времени выполнения «3»: возврат без GoSub

Неверный код

```
Sub DoSomething()
    GoSub DoThis
DoThis:
    Debug.Print "Hi!"
    Return
End Sub
```

Почему это не работает?

Выполнение входит в процедуру `DoSomething` , переходит на метку `DoThis` , печатает «Привет!». на выход отладки, *возвращается* к инструкции сразу после вызова `GoSub` , печатает «Привет!». снова, а затем встречает оператор `Return` , но до сих пор некуда *возвращаться* , потому что мы не попали сюда с инструкцией `GoSub` .

Правильный код

```
Sub DoSomething()
    GoSub DoThis
    Exit Sub
DoThis:
    Debug.Print "Hi!"
    Return
End Sub
```

Почему это работает?

Представляя инструкцию `Exit Sub` *перед* `DoThis` линии `DoThis` , мы `DoThis` подпрограмму `DoThis`

от остальной части тела процедуры - единственный способ выполнить подпрограмму `DoThis` - через прыжок `GoSub` .

Другие примечания

`GoSub` / `Return` устарел, и его следует избегать в пользу фактических вызовов процедур. Процедура не должна содержать подпрограмм, кроме обработчиков ошибок.

Это очень похоже на ошибку времени выполнения «20»: возобновить без ошибок ; в обеих ситуациях решение состоит в том, чтобы гарантировать, что *нормальный путь выполнения* не может войти в подпрограмму (обозначенную меткой строки) без явного перехода (предполагая, что `On Error GoTo` считается *явным скачком*).

Ошибка времени выполнения «6»: переполнение

неверный код

```
Sub DoSomething()
    Dim row As Integer
    For row = 1 To 100000
        'do stuff
    Next
End Sub
```

Почему это не работает?

Тип данных `Integer` - это 16-разрядное целое число со знаком с максимальным значением 32 767; присваивая его чему-либо большему, чем это приведет к *переполнению* типа и повышению этой ошибки.

Правильный код

```
Sub DoSomething()
    Dim row As Long
    For row = 1 To 100000
        'do stuff
    Next
End Sub
```

Почему это работает?

Вместо этого вместо этого используется `Long` (32-разрядное) целое число, теперь мы можем создать цикл, который выполняет итерацию более 32 767 раз, не переполняя тип переменной счетчика.

Другие примечания

Дополнительные сведения см. В разделе Типы данных и лимиты .

Ошибка времени выполнения «9»: подстрочный индекс

неверный код

```
Sub DoSomething()
    Dim foo(1 To 10)
    Dim i As Long
    For i = 1 To 100
        foo(i) = i
    Next
End Sub
```

Почему это не работает?

`foo` - массив, содержащий 10 элементов. Когда счетчик циклов `i` достигает значения 11, `foo(i)` *выходит за пределы диапазона* . Эта ошибка возникает всякий раз, когда доступ к массиву или коллекции осуществляется с индексом, который не существует в этом массиве или коллекции.

Правильный код

```
Sub DoSomething()
    Dim foo(1 To 10)
    Dim i As Long
    For i = LBound(foo) To UBound(foo)
        foo(i) = i
    Next
End Sub
```

Почему это работает?

Используйте функции `LBound` и `UBound` для определения нижней и верхней границ массива, соответственно.

Другие примечания

Когда индекс является строкой, например `ThisWorkbook.Worksheets("I don't exist")` , эта ошибка означает, что предоставленное имя не существует в запрошенной коллекции.

Фактическая ошибка является специфичной для реализации; `Collection` будет повышена ошибка во время выполнения 5 «Неверный вызов или аргумент процедуры»:

```
Sub RaisesRunTimeError5()
    Dim foo As New Collection
    foo.Add "foo", "foo"
    Debug.Print foo("bar")
End Sub
```

Ошибка времени выполнения «13»: несоответствие типов

неверный код

```
Public Sub DoSomething()
    DoSomethingElse "42?"
End Sub

Private Sub DoSomethingElse(foo As Date)
'    Debug.Print MonthName(Month(foo))
End Sub
```

Почему это не работает?

VBA пытается очень тяжело преобразовать `"42?"` аргумент в значение `Date` . Когда он терпит неудачу, вызов `DoSomethingElse` не может быть выполнен, потому что VBA не знает, какую дату передать, поэтому он вызывает *несоответствие типа* ошибки 13 во время выполнения, поскольку тип аргумента не соответствует ожидаемому типу (и может 'также неявно преобразуется).

Правильный код

```
Public Sub DoSomething()
    DoSomethingElse Now
End Sub

Private Sub DoSomethingElse(foo As Date)
'    Debug.Print MonthName(Month(foo))
End Sub
```

Почему это работает?

`Date` аргумент `Date` процедуре, которая ожидает параметр `Date` , вызов может быть успешным.

Ошибка времени выполнения '91': переменная объекта или с не заданной переменной блока

неверный код

```
Sub DoSomething()
    Dim foo As Collection
    With foo
        .Add "ABC"
        .Add "XYZ"
    End With
End Sub
```

Почему это не работает?

Переменные объекта содержат *ссылку* , а ссылки должны быть *заданы* с помощью ключевого слова « `Set` . Эта ошибка возникает , когда вызов элемента выполнен на объекте, ссылка не является `Nothing` . В этом случае `foo` является ссылкой `Collection` , но он не инициализирован, поэтому ссылка содержит `Nothing` - и мы не можем вызвать `.Add` on `Nothing` .

Правильный код

```
Sub DoSomething()
    Dim foo As Collection
    Set foo = New Collection
    With foo
        .Add "ABC"
        .Add "XYZ"
    End With
End Sub
```

Почему это работает?

`.Add` объектной переменной допустимую ссылку с помощью ключевого слова `Set` , вызовы `.Add` успешными.

Другие примечания

Часто функция или свойство может возвращать ссылку на объект - распространенным примером является метод Excel `Range.Find` , который возвращает объект `Range` :

```
Dim resultRow As Long
resultRow = SomeSheet.Cells.Find("Something").Row
```

Однако функция может очень хорошо вернуть `Nothing` (если поисковый `.Row` не найден), поэтому вполне вероятно, что `.Row` вызова `.Row` не выполняется.

Перед вызовом членов объекта убедитесь, что ссылка задана с условием `If Not xxxx Is Nothing` :

```
Dim result As Range
```

```
Set result = SomeSheet.Cells.Find("Something")

Dim resultRow As Long
If Not result Is Nothing Then resultRow = result.Row
```

Ошибка времени выполнения «20»: возобновить без ошибок

неверный код

```
Sub DoSomething()
    On Error GoTo CleanFail
    DoSomethingElse

CleanFail:
    Debug.Print Err.Number
    Resume Next
End Sub
```

Почему это не работает?

Если процедура `DoSomethingElse` вызывает ошибку, выполнение переходит к `CleanFail` линии `CleanFail` , печатает номер ошибки, а команда `Resume Next` возвращается к инструкции, которая сразу же следует за строкой, в которой произошла ошибка, которая в этом случае является `Debug.Print` инструкция: подпрограмма обработки ошибок выполняется без контекста ошибки, и когда достигается команда `Resume Next` , возникает ошибка 20 времени выполнения, потому что возобновить ее некуда.

Правильный код

```
Sub DoSomething()
    On Error GoTo CleanFail
    DoSomethingElse

    Exit Sub
CleanFail:
    Debug.Print Err.Number
    Resume Next
End Sub
```

Почему это работает?

Представляя инструкцию `Exit Sub` перед `CleanFail` линии `CleanFail` , мы `CleanFail` подпрограмму обработки ошибок `CleanFail` от остальной части тела процедуры - единственный способ выполнить подпрограмму обработки ошибок - с помощью скачка `On Error` ; поэтому путь выполнения не доходит до инструкции `Resume` за пределами контекста ошибки, что позволяет избежать ошибки времени выполнения 20.

Другие примечания

Это очень похоже на ошибку времени выполнения «3»: возврат без GoSub ; в обеих ситуациях решение состоит в том, чтобы гарантировать, что *нормальный путь выполнения* не может войти в подпрограмму (обозначенную меткой строки) без явного перехода (предполагая, что `On Error GoTo` считается *явным скачком*).

Прочитайте Ошибки времени выполнения VBA онлайн:
https://riptutorial.com/ru/vba/topic/8917/ошибки-времени-выполнения-vba

глава 26: Передача аргументов ByRef или ByVal

Вступление

Модификаторы `ByRef` и `ByVal` являются частью сигнатуры процедуры и указывают, как передается аргумент процедуре. В VBA параметр передается `ByRef` если не указано иное (т.е. `ByRef` неявно, если отсутствует).

Примечание. Во многих других языках программирования (включая VB.NET) параметры неявно передаются по значению, если не указан модификатор: подумайте о том, чтобы указать модификаторы `ByRef` явно, чтобы избежать возможной путаницы.

замечания

Передача массивов

Массивы **должны** передаваться по ссылке. Этот код компилируется, но вызывает ошибку времени выполнения 424 «Object Required»:

```
Public Sub Test()
    DoSomething Array(1, 2, 3)
End Sub

Private Sub DoSomething(ByVal foo As Variant)
    foo.Add 42
End Sub
```

Этот код не компилируется:

```
Private Sub DoSomething(ByVal foo() As Variant) 'ByVal is illegal for arrays
    foo.Add 42
End Sub
```

Examples

Передача простых переменных ByRef And ByVal

Передача `ByRef` или `ByVal` указывает, `ByVal` ли фактическое значение аргумента `CalledProcedure` `CallingProcedure` , или же ссылка (называемая указателем на некоторых других языках) передается `CalledProcedure` .

Если аргумент передан `ByRef` , адрес памяти аргумента передается в `CalledProcedure` и

любая модификация этого параметра `CalledProcedure` выполняется в значение `CallingProcedure` .

Если аргумент передается `ByVal` , фактическое значение, а не ссылка на переменную, передается в `CalledProcedure` .

Простой пример будет ясно иллюстрировать это:

```
Sub CalledProcedure(ByRef X As Long, ByVal Y As Long)
    X = 321
    Y = 654
End Sub

Sub CallingProcedure()
    Dim A As Long
    Dim B As Long
    A = 123
    B = 456

    Debug.Print "BEFORE CALL => A: " & CStr(A), "B: " & CStr(B)
    ''Result : BEFORE CALL => A: 123 B: 456

    CalledProcedure X:=A, Y:=B

    Debug.Print "AFTER CALL =  A: " & CStr(A), "B: " & CStr(B)
    ''Result : AFTER CALL => A: 321 B: 456
End Sub
```

Другой пример:

```
Sub Main()
    Dim IntVarByVal As Integer
    Dim IntVarByRef As Integer

    IntVarByVal = 5
    IntVarByRef = 10

    SubChangeArguments IntVarByVal, IntVarByRef '5 goes in as a "copy". 10 goes in as a
reference
    Debug.Print "IntVarByVal: " & IntVarByVal 'prints 5 (no change made by SubChangeArguments)
    Debug.Print "IntVarByRef: " & IntVarByRef 'prints 99 (the variable was changed in
SubChangeArguments)
End Sub

Sub SubChangeArguments(ByVal ParameterByVal As Integer, ByRef ParameterByRef As Integer)
    ParameterByVal = ParameterByVal + 2 ' 5 + 2 = 7 (changed only inside this Sub)
    ParameterByRef = ParameterByRef + 89 ' 10 + 89 = 99 (changes the IntVarByRef itself - in
the Main Sub)
End Sub
```

ByRef

Модификатор по умолчанию

Если для параметра не указан модификатор, этот параметр неявно передается по ссылке.

```
Public Sub DoSomething1(foo As Long)
End Sub
```

```
Public Sub DoSomething2(ByRef foo As Long)
End Sub
```

Параметр `foo` передается `ByRef` как в `DoSomething1` и `DoSomething2` .

> **Осторожно!** Если вы приходите в VBA с опытом работы на других языках, это, скорее всего, совершенно противоположное поведение с тем, к которому вы привыкли. Во многих других языках программирования (включая VB.NET) неявный / по умолчанию модификатор передает параметры по значению.

Передача по ссылке

- Когда *значение* передается `ByRef` , процедура получает **ссылку** на значение.

```
Public Sub Test()
    Dim foo As Long
    foo = 42
    DoSomething foo
    Debug.Print foo
End Sub

Private Sub DoSomething(ByRef foo As Long)
    foo = foo * 2
End Sub
```

Вызов вышеуказанных выходов `Test` процедуры 84. `DoSomething` присваивается `foo` и получает *ссылку* на значение, и поэтому работает с тем же адресом памяти, что и вызывающий.

- Когда *ссылка* передается `ByRef` , процедура получает **ссылку** на указатель.

```
Public Sub Test()
    Dim foo As Collection
    Set foo = New Collection
    DoSomething foo
    Debug.Print foo.Count
End Sub

Private Sub DoSomething(ByRef foo As Collection)
    foo.Add 42
    Set foo = Nothing
End Sub
```

Вышеприведенный код повышает ошибку 91 во время выполнения , поскольку вызывающий абонент вызывает член `Count` объекта, который больше не существует,

поскольку `DoSomething` получил *ссылку* на указатель объекта и назначил его `Nothing` перед возвратом.

Принуждение ByVal на сайте вызова

Используя круглые скобки на сайте вызова, вы можете переопределить `ByRef` и принудительно передать аргумент `ByVal` :

```
Public Sub Test()
    Dim foo As Long
    foo = 42
    DoSomething (foo)
    Debug.Print foo
End Sub

Private Sub DoSomething(ByRef foo As Long)
    foo = foo * 2
End Sub
```

Вышеуказанные выходы кода 42, независимо от того, указан ли `ByRef` неявно или явно.

> **Осторожно!** Из-за этого, используя посторонние круглые скобки в процедурных вызовах, можно легко ввести ошибки. Обратите внимание на пробелы между именем процедуры и списком аргументов:
>
> ```
> bar = DoSomething(foo) 'function call, no whitespace; parens are part of args list
> DoSomething (foo) 'procedure call, notice whitespace; parens are NOT part of args
> list
> DoSomething foo 'procedure call does not force the foo parameter to be ByVal
> ```

ByVal

Передача по значению

- Когда *значение* передается `ByVal` , процедура получает **копию** значения.

```
Public Sub Test()
    Dim foo As Long
    foo = 42
    DoSomething foo
    Debug.Print foo
End Sub

Private Sub DoSomething(ByVal foo As Long)
    foo = foo * 2
End Sub
```

Вызов вышеуказанных выходов `Test` процедуры 42. `DoSomething` присваивается `foo` и получает **копию** значения. Копия умножается на 2, а затем отменяется, когда

процедура завершается; копия вызывающего абонента никогда не изменялась.

- Когда *ссылка* передается `ByVal` , процедура получает **копию** указателя.

```
Public Sub Test()
    Dim foo As Collection
    Set foo = New Collection
    DoSomething foo
    Debug.Print foo.Count
End Sub

Private Sub DoSomething(ByVal foo As Collection)
    foo.Add 42
    Set foo = Nothing
End Sub
```

Вызов вышеуказанных результатов `Test` процедуры 1. `DoSomething` предоставляется `foo` и получает *копию* **указателя** на объект `Collection` . Поскольку переменная объекта `foo` в области `Test` указывает на один и тот же объект, добавление элемента в `DoSomething` добавляет элемент к одному и тому же объекту. Поскольку это *копия* указателя, установка ссылки на `Nothing` не влияет на собственную копию вызывающего абонента.

Прочитайте Передача аргументов ByRef или ByVal онлайн:
https://riptutorial.com/ru/vba/topic/7363/передача-аргументов-byref-или-byval

глава 27: Подстроки

замечания

VBA имеет встроенные функции для извлечения определенных частей строк, в том числе:

- `Left` / `Left$`
- `Right` / `Right$`
- `Mid` / `Mid$`
- `Trim` / `Trim$`

Чтобы избежать неявного преобразования типа onverhead (и, следовательно, для повышения производительности), используйте $ -suffixed версию функции, когда строковая переменная передается функции, и / или если результат функции присваивается строковой переменной.

Передача значения параметра `Null` функции $ -suffixed приведет к возникновению ошибки времени выполнения («недопустимое использование null») - это особенно актуально для кода, связанного с базой данных.

Examples

Используйте Left или Left $, чтобы получить 3 левых символа в строке

```
Const baseString As String = "Foo Bar"

Dim leftText As String
leftText = Left$(baseString, 3)
'leftText = "Foo"
```

Используйте правую или правую $, чтобы получить 3 правильных символа в строке

```
Const baseString As String = "Foo Bar"
Dim rightText As String
rightText = Right$(baseString, 3)
'rightText = "Bar"
```

Используйте Mid или Mid $ для получения определенных символов из строки

```
Const baseString As String = "Foo Bar"

'Get the string starting at character 2 and ending at character 6
```

```
Dim midText As String
midText = Mid$(baseString, 2, 5)
'midText = "oo Ba"
```

Используйте Trim для получения копии строки без каких-либо ведущих или конечных пробелов

```
'Trim the leading and trailing spaces in a string
Const paddedText As String = "    Foo Bar    "
Dim trimmedText As String
trimmedText = Trim$(paddedText)
'trimmedText = "Foo Bar"
```

Прочитайте Подстроки онлайн: https://riptutorial.com/ru/vba/topic/3481/подстроки

глава 28: Поиск в строках для наличия подстрок

замечания

Когда вам нужно проверить наличие или позицию подстроки внутри строки, VBA предлагает функции `InStr` и `InStrRev` которые возвращают позицию символа подстроки в строке, если она присутствует.

Examples

Используйте InStr, чтобы определить, содержит ли строка подстроку

```
Const baseString As String = "Foo Bar"
Dim containsBar As Boolean

'Check if baseString contains "bar" (case insensitive)
containsBar = InStr(1, baseString, "bar", vbTextCompare) > 0
'containsBar = True

'Check if baseString contains bar (case insensitive)
containsBar = InStr(1, baseString, "bar", vbBinaryCompare) > 0
'containsBar = False
```

Используйте InStr, чтобы найти позицию первого экземпляра подстроки

```
Const baseString As String = "Foo Bar"
Dim containsBar As Boolean

Dim posB As Long
posB = InStr(1, baseString, "B", vbBinaryCompare)
'posB = 5
```

Используйте InStrRev, чтобы найти позицию последнего экземпляра подстроки

```
Const baseString As String = "Foo Bar"
Dim containsBar As Boolean

'Find the position of the last "B"
Dim posX As Long
'Note the different number and order of the paramters for InStrRev
posX = InStrRev(baseString, "X", -1, vbBinaryCompare)
'posX = 0
```

Прочитайте Поиск в строках для наличия подстрок онлайн:

https://riptutorial.com/ru/vba/topic/3480/поиск-в-строках-для-наличия-подстрок

глава 29: Пользовательские формы

Examples

Лучшие практики

`UserForm` - это модуль класса с дизайнером и **экземпляром по умолчанию** . Доступ к *дизайнеру* можно получить, нажав `Shift` + `F7` при просмотре *кода* , а доступ к *коду* можно получить, нажав `F7` во время просмотра *дизайнера* .

Работайте с новым экземпляром каждый раз.

Будучи *модулем класса* , форма является, таким образом, *планом* для *объекта* . Поскольку форма может содержать состояние и данные, лучше работать с новым *экземпляром* класса, а не с по умолчанию / глобальным:

```
With New UserForm1
    .Show vbModal
    If Not .IsCancelled Then
        '...
    End If
End With
```

Вместо:

```
UserForm1.Show vbModal
If Not UserForm1.IsCancelled Then
    '...
End If
```

Работа с экземпляром по умолчанию может привести к тонким ошибкам, когда форма закрыта красной кнопкой «X» и / или когда `Unload Me` используется в коде.

Реализуйте логику в другом месте.

Форма должна касаться только *презентации* : кнопка `Click` handler, которая подключается к базе данных и запускает параметризованный запрос на основе ввода пользователем, **делает слишком много вещей** .

Вместо этого внедрите *прикладную логику* в код, который отвечает за отображение формы или даже лучше в выделенных модулях и процедурах.

Напишите код таким образом, чтобы UserForm всегда отвечал за знание того, как отображать и собирать данные: откуда поступают данные или что происходит с данными

после этого, не является проблемой.

Абонент не должен беспокоиться о контроле.

Создайте хорошо определенную *модель* для формы, с которой можно работать, либо в своем собственном выделенном модуле класса, либо инкапсулироваться внутри самого кода кода - выставлять *модель* с процедурами `Property Get` и работать с этим кодом клиента: это делает форма *абстракции* над элементами управления и их подробные детали, подвергая только соответствующие данные клиенту.

Это означает, что код выглядит следующим образом:

```
With New UserForm1
    .Show vbModal
    If Not .IsCancelled Then
        MsgBox .Message, vbInformation
    End If
End With
```

Вместо этого:

```
With New UserForm1
    .Show vbModal
    If Not .IsCancelled Then
        MsgBox .txtMessage.Text, vbInformation
    End If
End With
```

Обработать событие QueryClose.

Обычно формы имеют кнопку « `Закрыть`» , а в подсказках / диалоговом окне есть кнопки « `OK`» и « `Отмена`» ; пользователь может закрыть форму, используя *блок управления* формы (красная кнопка «X»), которая по умолчанию уничтожает экземпляр формы (еще одна веская причина *работать с новым экземпляром каждый раз*).

```
With New UserForm1
    .Show vbModal
    If Not .IsCancelled Then 'if QueryClose isn't handled, this can raise a runtime error.
        '...
    End With
End With
```

Самый простой способ обработать событие `QueryClose` - установить для параметра `Cancel` значение `True` , а затем *скрыть* форму, а не *закрывать* ее:

```
Private Sub UserForm_QueryClose(Cancel As Integer, CloseMode As Integer)
    Cancel = True
    Me.Hide
```

```
End Sub
```

Таким образом, кнопка «X» никогда не уничтожит экземпляр, и вызывающий может безопасно получить доступ ко всем открытым членам.

Скрыть, не закрывать.

Код, который создает объект, должен нести ответственность за его уничтожение: ответственность за разгрузку и завершение работы лежит не на форме.

Избегайте использования `Unload Me` в коде кода. Вызовите `Me.Hide` вместо этого, чтобы вызывающий код все еще мог использовать объект, который он создал, когда форма закрывается.

Назовите вещи.

Используйте окно *свойств* (`F4`) для тщательного определения каждого элемента управления в форме. Имя элемента управления используется в коде, так что, если вы не используете инструмент рефакторинга, который может справиться с этим, **переименование элемента управления приведет к поломке кода** - так что гораздо проще делать все правильно, в первую очередь, чем пытаться чтобы точно определить, какой из 20 текстовых `TextBox12` означает.

Традиционно элементы управления UserForm называются префиксами венгерского стиля:

- `lblUserName` для `lblUserName` управления `Label` , указывающего имя пользователя.
- `txtUserName` для `txtUserName` управления `TextBox` где пользователь может ввести имя пользователя.
- `cboUserName` для управления `ComboBox` где пользователь может ввести или выбрать имя пользователя.
- `lstUserName` для `lstUserName` управления `ListBox` где пользователь может выбрать имя пользователя.
- `btnOk` или `cmdOk` для управления `Button` с надписью «Ok».

Проблема заключается в том, что когда, например, пользовательский интерфейс изменен и `ComboBox` изменяется на `ListBox` , имя необходимо изменить, чтобы отразить новый тип управления: лучше называть элементы управления для того, что они представляют, а не после их типа управления - *отделить* кода из пользовательского интерфейса, насколько это возможно.

- `UserNameLabel` для метки только для чтения, которая указывает имя пользователя.
- `UserNameInput` для `UserNameInput` управления, в котором пользователь может ввести или выбрать имя пользователя.
- `OkButton`

для командной кнопки с надписью «Ok».

Какой бы стиль ни выбрали, все лучше, чем оставлять все элементы управления по умолчанию. Согласованность в стиле именования тоже идеальна.

Обработка QueryClose

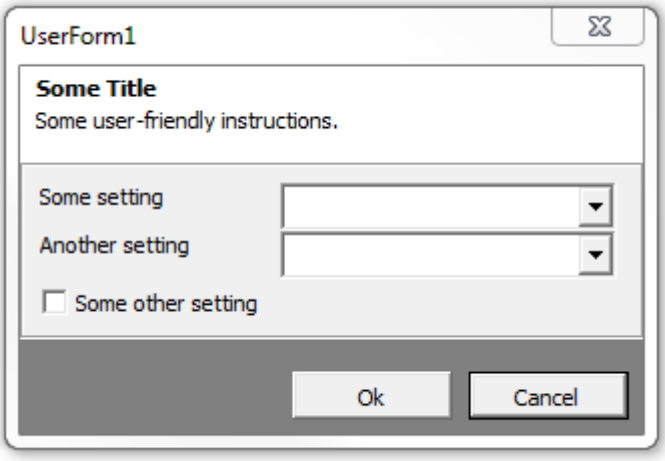
Событие `QueryClose` возникает всякий раз, когда форма закрывается, независимо от того, выполняется ли это с помощью действия пользователя или программно. Параметр `CloseMode` содержит `VbQueryClose` перечисления `VbQueryClose` которое указывает, как форма была закрыта:

постоянная	Описание	Значение
`vbFormControlMenu`	Форма закрывается в ответ на действие пользователя	0
`vbFormCode`	Форма закрывается в ответ на оператор `Unload`	1
`vbAppWindows`	Сеанс Windows завершается	2
`vbAppTaskManager`	Диспетчер задач Windows закрывает приложение-хост	3
`vbFormMDIForm`	Не поддерживается в VBA	4

Для лучшей читаемости лучше использовать эти константы, а не использовать их значение напрямую.

Отмена пользовательской формы

Учитывая форму с кнопкой « `Отмена`»



Кодировка кода формы может выглядеть так:

```
Option Explicit
Private Type TView
    IsCancelled As Boolean
    SomeOtherSetting As Boolean
    'other properties skipped for brievety
End Type
Private this As TView

Public Property Get IsCancelled() As Boolean
    IsCancelled = this.IsCancelled
End Property

Public Property Get SomeOtherSetting() As Boolean
    SomeOtherSetting = this.SomeOtherSetting
End Property

'...more properties...

Private Sub SomeOtherSettingInput_Change()
    this.SomeOtherSetting = CBool(SomeOtherSettingInput.Value)
End Sub

Private Sub OkButton_Click()
    Me.Hide
End Sub

Private Sub CancelButton_Click()
    this.IsCancelled = True
    Me.Hide
End Sub

Private Sub UserForm_QueryClose(Cancel As Integer, CloseMode As Integer)
    If CloseMode = VbQueryClose.vbFormControlMenu Then
        Cancel = True
        this.IsCancelled = True
        Me.Hide
    End If
End Sub
```

Затем вызывающий код может отображать форму и знать, была ли она отменена:

```
Public Sub DoSomething()
    With New UserForm1
        .Show vbModal
        If .IsCancelled Then Exit Sub
        If .SomeOtherSetting Then
            'setting is enabled
        Else
            'setting is disabled
        End If
    End With
End Sub
```

Свойство `IsCancelled` возвращает `True` при нажатии кнопки « `Отмена`» или когда пользователь закрывает форму с помощью *блока управления* .

Прочитайте Пользовательские формы онлайн: https://riptutorial.com/ru/vba/topic/5351/пользовательские-формы

глава 30: Преобразование других типов в строки

замечания

VBA будет неявно преобразовывать некоторые типы в строку по мере необходимости и без какой-либо дополнительной работы со стороны программиста, но VBA также предоставляет ряд явных функций преобразования строк, и вы также можете написать свой собственный.

Три из наиболее часто используемых функций - `CStr` , `Format` и `StrConv` .

Examples

Используйте CStr для преобразования числового типа в строку

```
Const zipCode As Long = 10012
Dim zipCodeText As String
'Convert the zipCode number to a string of digit characters
zipCodeText = CStr(zipCode)
'zipCodeText = "10012"
```

Использовать формат для преобразования и форматирования числового типа в виде строки

```
Const zipCode As long = 10012
Dim zeroPaddedNumber As String
zeroPaddedZipCode = Format(zipCode, "00000000")
'zeroPaddedNumber = "00010012"
```

Используйте StrConv для преобразования байтового массива однобайтных символов в строку

```
'Declare an array of bytes, assign single-byte character codes, and convert to a string
Dim singleByteChars(4) As Byte
singleByteChars(0) = 72
singleByteChars(1) = 101
singleByteChars(2) = 108
singleByteChars(3) = 108
singleByteChars(4) = 111
Dim stringFromSingleByteChars As String
stringFromSingleByteChars = StrConv(singleByteChars, vbUnicode)
'stringFromSingleByteChars = "Hello"
```

Неявно преобразовать массив байтов многобайтовых символов в строку

```
'Declare an array of bytes, assign multi-byte character codes, and convert to a string
Dim multiByteChars(9) As Byte
multiByteChars(0) = 87
multiByteChars(1) = 0
multiByteChars(2) = 111
multiByteChars(3) = 0
multiByteChars(4) = 114
multiByteChars(5) = 0
multiByteChars(6) = 108
multiByteChars(7) = 0
multiByteChars(8) = 100
multiByteChars(9) = 0

Dim stringFromMultiByteChars As String
stringFromMultiByteChars = multiByteChars
'stringFromMultiByteChars = "World"
```

Прочитайте Преобразование других типов в строки онлайн:
https://riptutorial.com/ru/vba/topic/3467/преобразование-других-типов-в-строки

глава 31: Процедурные вызовы

Синтаксис

- Идентификатор_имя [*аргументы*]
- Идентификатор вызова [[*аргументы*]]
- [Let | Set] *expression* = ИдентификаторName [*(аргументы)*]
- [Let | Set] ИдентификаторName [*(arguments)*] = *выражение*

параметры

параметр	Информация
ИмяИдентификатора	Имя процедуры для вызова.
аргументы	Список аргументов, разделенных запятыми, которые должны быть переданы процедуре.

замечания

Первые два синтаксиса предназначены для вызова процедур `Sub` ; обратите внимание, что первый синтаксис не содержит круглых скобок.

См. Это сбивает с толку. Почему бы просто не использовать круглые скобки? для подробного объяснения различий между двумя первыми двумя синтаксисами.

Третий синтаксис предназначен для вызова процедур `Function` и `Property Get` ; когда есть параметры, скобки всегда обязательны. Ключевое слово `Let` является необязательным при назначении *значения* , но ключевое слово `Set` **требуется** при назначении *ссылки* .

Четвертый синтаксис предназначен для вызова процедур `Property Let` и `Property Set` ; *`expression`* в правой части присваивания передается параметру значения свойства.

Examples

Синтаксис неявного вызова

```
ProcedureName
ProcedureName argument1, argument2
```

Вызовите процедуру по ее имени без каких-либо круглых скобок.

Кронштейн

Ключевое слово `Call` требуется только в одном случае:

```
Call DoSomething : DoSomethingElse
```

`DoSomething` и `DoSomethingElse` - вызываемые процедуры. Если ключевое слово `Call` было удалено, то `DoSomething` будет анализироваться как *метка строки,* а не вызов процедуры, который нарушит код:

```
DoSomething: DoSomethingElse 'only DoSomethingElse will run
```

Возвращаемые значения

Чтобы получить результат вызова процедуры (например, `Function` или процедуры получения `Property Get`), поместите вызов в правую часть задания:

```
result = ProcedureName
result = ProcedureName(argument1, argument2)
```

Скобки должны присутствовать, если есть параметры. Если процедура не имеет параметров, скобки являются избыточными.

Это смущает. Почему бы просто не использовать круглые скобки?

Круглые скобки используются для включения аргументов *вызовов функций* .
Использование их для *вызовов процедур* может вызвать непредвиденные проблемы.

Поскольку они могут вводить ошибки, как во время выполнения, передавая возможно непреднамеренное значение для процедуры, так и во время компиляции, просто будучи недействительным синтаксисом.

Во время выполнения

Резервные круглые скобки могут вводить ошибки. Учитывая процедуру, которая принимает ссылку на объект как параметр ...

```
Sub DoSomething(ByRef target As Range)
End Sub
```

... и вызывается с круглыми скобками:

```
DoSomething (Application.ActiveCell) 'raises an error at runtime
```

Это вызовет ошибку времени выполнения «Требуемый объект» # 424. Другие ошибки возможны и в других обстоятельствах: здесь ссылка на объект `Application.ActiveCell Range` *оценивается* и передается по значению **независимо** от сигнатуры процедуры, указывающей, что `target` будет передана `ByRef` . Фактическое значение, переданное `ByVal` в `DoSomething` в приведенном выше фрагменте, - `Application.ActiveCell.Value` .

Скобки заставляют VBA оценивать значение выражения в квадратных скобках и передавать результат `ByVal` вызываемой процедуре. Когда тип оцениваемого результата не соответствует ожидаемому типу процедуры и не может быть неявно преобразован, возникает ошибка времени выполнения.

Во время компиляции

Этот код не сможет скомпилировать:

```
MsgBox ("Invalid Code!", vbCritical)
```

Поскольку выражение `("Invalid Code!", vbCritical)` не может быть *оценено* до значения.

Это будет скомпилировано и работает:

```
MsgBox ("Invalid Code!"), (vbCritical)
```

Но определенно будет выглядеть глупо. Избегайте избыточных скобок.

Синтаксис явного вызова

```
Call ProcedureName
Call ProcedureName(argument1, argument2)
```

Явный синтаксис вызова требует ключевого слова `Call` и круглых скобок вокруг списка аргументов; круглые скобки являются избыточными, если нет параметров. Этот синтаксис был устаревшим, когда в VB был добавлен более современный неявный синтаксис вызова.

Необязательные аргументы

Некоторые процедуры имеют необязательные аргументы. Необязательные аргументы всегда появляются после необходимых аргументов, но процедура может быть вызвана без них.

Например, если функция, `ProcedureName` должна иметь два обязательных аргумента (`argument1` , `argument2`) и один необязательный аргумент `optArgument3` , его можно было бы назвать как минимум четырьмя способами:

```
' Without optional argument
result = ProcedureName("A", "B")

' With optional argument
result = ProcedureName("A", "B", "C")

' Using named arguments (allows a different order)
result = ProcedureName(optArgument3:="C", argument1:="A", argument2:="B")

' Mixing named and unnamed arguments
result = ProcedureName("A", "B", optArgument3:="C")
```

Структура вызываемого здесь заголовка функции будет выглядеть примерно так:

```
Function ProcedureName(argument1 As String, argument2 As String, Optional optArgument3 As
String) As String
```

Ключевое слово `Optional` указывает, что этот аргумент можно опустить. Как уже упоминалось ранее, любые необязательные аргументы, введенные в заголовок, **должны** появляться в конце после любых необходимых аргументов.

Вы также можете *указать* значение по *умолчанию* для аргумента в том случае, если значение не передается функции:

```
Function ProcedureName(argument1 As String, argument2 As String, Optional optArgument3 As
String = "C") As String
```

В этой функции, если аргумент для `c` не указан, значение будет по умолчанию равно `"C"` . Если задано значение , это переопределит значение по умолчанию.

Прочитайте Процедурные вызовы онлайн: https://riptutorial.com/ru/vba/topic/1179/процедурные-вызовы

глава 32: Работа с ADO

замечания

Примеры, показанные в этом разделе, используют раннее связывание для ясности и требуют ссылки на библиотеку xx Library ActiveX Data ActiveX. Они могут быть преобразованы в позднее связывание, заменив строго типизированные ссылки на `Object` и заменяя создание объекта с помощью `New` с `CreateObject` где это необходимо.

Examples

Подключение к источнику данных

Первым шагом в доступе к источнику данных через ADO является создание объекта ADO `Connection` . Обычно это делается с использованием строки подключения для указания параметров источника данных, хотя также можно открыть соединение DSN, передав DSN, идентификатор пользователя и пароль в метод `.Open` .

Обратите внимание, что DSN не требуется для подключения к источнику данных через ADO - любой источник данных, у которого есть поставщик ODBC, может быть подключен к соответствующей строке соединения. Хотя конкретные строки подключения для разных поставщиков не входят в сферу применения этой темы, ConnectionStrings.com является отличной ссылкой для поиска соответствующей строки для вашего провайдера.

```
Const SomeDSN As String = "DSN=SomeDSN;Uid=UserName;Pwd=MyPassword;"

Public Sub Example()
    Dim database As ADODB.Connection
    Set database = OpenDatabaseConnection(SomeDSN)
    If Not database Is Nothing Then
        '... Do work.
        database.Close          'Make sure to close all database connections.
    End If
End Sub

Public Function OpenDatabaseConnection(ConnString As String) As ADODB.Connection
    On Error GoTo Handler
    Dim database As ADODB.Connection
    Set database = New ADODB.Connection

    With database
        .ConnectionString = ConnString
        .ConnectionTimeout = 10             'Value is given in seconds.
        .Open
    End With

    OpenDatabaseConnection = database

    Exit Function
```

```
Handler:
    Debug.Print "Database connection failed. Check your connection string."
End Function
```

Обратите внимание, что пароль базы данных включен в строку соединения в приведенном выше примере только для ясности. Наилучшая практика будет диктовать ***не*** хранение паролей базы данных в коде. Это может быть выполнено путем ввода пароля через пользовательский ввод или с помощью проверки подлинности Windows.

Получение записей с запросом

Запросы могут выполняться двумя способами, оба из которых возвращают объект ADO `Recordset` который представляет собой набор возвращенных строк. Обратите внимание, что в обоих примерах ниже используется функция `OpenDatabaseConnection` из `OpenDatabaseConnection` « Создание соединения с источником данных» для краткости. Помните, что синтаксис SQL, переданный источнику данных, специфичен для провайдера.

Первый метод - передать инструкцию SQL непосредственно объекту Connection и является самым простым методом для выполнения простых запросов:

```
Public Sub DisplayDistinctItems()
    On Error GoTo Handler
    Dim database As ADODB.Connection
    Set database = OpenDatabaseConnection(SomeDSN)

    If Not database Is Nothing Then
        Dim records As ADODB.Recordset
        Set records = database.Execute("SELECT DISTINCT Item FROM Table")
        'Loop through the returned Recordset.
        Do While Not records.EOF      'EOF is false when there are more records.
            'Individual fields are indexed either by name or 0 based ordinal.
            'Note that this is using the default .Fields member of the Recordset.
            Debug.Print records("Item")
            'Move to the next record.
            records.MoveNext
        Loop
    End If
CleanExit:
    If Not records Is Nothing Then records.Close
    If Not database Is Nothing And database.State = adStateOpen Then
        database.Close
    End If
    Exit Sub
Handler:
    Debug.Print "Error " & Err.Number & ": " & Err.Description
    Resume CleanExit
End Sub
```

Второй способ - создать объект `Command` ADO для запроса, который вы хотите выполнить. Для этого требуется немного больше кода, но это необходимо для использования параметризованных запросов:

```
Public Sub DisplayDistinctItems()
    On Error GoTo Handler
    Dim database As ADODB.Connection
    Set database = OpenDatabaseConnection(SomeDSN)

    If Not database Is Nothing Then
        Dim query As ADODB.Command
        Set query = New ADODB.Command
        'Build the command to pass to the data source.
        With query
            .ActiveConnection = database
            .CommandText = "SELECT DISTINCT Item FROM Table"
            .CommandType = adCmdText
        End With
        Dim records As ADODB.Recordset
        'Execute the command to retrieve the recordset.
        Set records = query.Execute()

        Do While Not records.EOF
            Debug.Print records("Item")
            records.MoveNext
        Loop
    End If
CleanExit:
    If Not records Is Nothing Then records.Close
    If Not database Is Nothing And database.State = adStateOpen Then
        database.Close
    End If
    Exit Sub
Handler:
    Debug.Print "Error " & Err.Number & ": " & Err.Description
    Resume CleanExit
End Sub
```

Обратите внимание, что команды, отправленные источнику данных, **уязвимы для SQL-инъекций** , преднамеренных или непреднамеренных. В общем случае запросы не должны создаваться путем объединения пользовательского ввода любого типа. Вместо этого они должны быть параметризованы (см. Создание параметризованных команд).

Выполнение нескалярных функций

Соединения ADO могут использоваться для выполнения практически любой функции базы данных, которую поставщик поддерживает через SQL. В этом случае не всегда необходимо использовать `Recordset` возвращаемый функцией `Execute` , хотя он может быть полезен для получения назначений ключей после операторов INSERT с помощью @@ Identity или аналогичных SQL-команд. Обратите внимание, что приведенный ниже пример использует функцию `OpenDatabaseConnection` из `OpenDatabaseConnection` « Создание соединения с источником данных» для краткости.

```
Public Sub UpdateTheFoos()
    On Error GoTo Handler
    Dim database As ADODB.Connection
    Set database = OpenDatabaseConnection(SomeDSN)
```

```
    If Not database Is Nothing Then
        Dim update As ADODB.Command
        Set update = New ADODB.Command
        'Build the command to pass to the data source.
        With update
            .ActiveConnection = database
            .CommandText = "UPDATE Table SET Foo = 42 WHERE Bar IS NULL"
            .CommandType = adCmdText
            .Execute        'We don't need the return from the DB, so ignore it.
        End With
    End If
CleanExit:
    If Not database Is Nothing And database.State = adStateOpen Then
        database.Close
    End If
    Exit Sub
Handler:
    Debug.Print "Error " & Err.Number & ": " & Err.Description
    Resume CleanExit
End Sub
```

Обратите внимание, что команды, отправленные источнику данных, **уязвимы для SQL-инъекций** , преднамеренных или непреднамеренных. В общем случае SQL-запросы не должны создаваться путем объединения пользовательских данных любого типа. Вместо этого они должны быть параметризованы (см. Создание параметризованных команд).

Создание параметризованных команд

Каждый раз, когда SQL, выполняемый через соединение ADO, должен содержать пользовательский ввод, считается оптимальной для его параметризации, чтобы минимизировать вероятность внедрения SQL. Этот метод также более читабельен, чем длинные конкатенации, и обеспечивает более надежный и поддерживаемый код (т. Е. С помощью функции, возвращающей массив `Parameter`).

В стандартном синтаксисе ODBC задаются параметры `?` «заполнители» в тексте запроса, а затем параметры добавляются к `Command` в том же порядке, что и в запросе.

Обратите внимание, что в приведенном ниже примере для краткости используется функция `OpenDatabaseConnection` из `OpenDatabaseConnection` « Создание соединения с источником данных» .

```
Public Sub UpdateTheFoos()
    On Error GoTo Handler
    Dim database As ADODB.Connection
    Set database = OpenDatabaseConnection(SomeDSN)

    If Not database Is Nothing Then
        Dim update As ADODB.Command
        Set update = New ADODB.Command
        'Build the command to pass to the data source.
        With update
            .ActiveConnection = database
            .CommandText = "UPDATE Table SET Foo = ? WHERE Bar = ?"
```

```
            .CommandType = adCmdText

            'Create the parameters.
            Dim fooValue As ADODB.Parameter
            Set fooValue = .CreateParameter("FooValue", adNumeric, adParamInput)
            fooValue.Value = 42

            Dim condition As ADODB.Parameter
            Set condition = .CreateParameter("Condition", adBSTR, adParamInput)
            condition.Value = "Bar"

            'Add the parameters to the Command
            .Parameters.Append fooValue
            .Parameters.Append condition
            .Execute
        End With
    End If
CleanExit:
    If Not database Is Nothing And database.State = adStateOpen Then
        database.Close
    End If
    Exit Sub
Handler:
    Debug.Print "Error " & Err.Number & ": " & Err.Description
    Resume CleanExit
End Sub
```

Примечание. В приведенном выше примере демонстрируется параметризованный оператор UPDATE, но любому оператору SQL могут быть заданы параметры.

Прочитайте Работа с ADO онлайн: https://riptutorial.com/ru/vba/topic/3578/работа-с-ado

глава 33: Работа с файлами и каталогами без использования FileSystemObject

замечания

`Scripting.FileSystemObject` гораздо более надежный, чем унаследованные методы в этом разделе. Это должно быть предпочтительным почти во всех случаях.

Examples

Определение наличия папок и файлов

файлы:

Чтобы определить, существует ли файл, просто передайте имя файла функции `Dir$` и проверьте, возвращает ли он результат. Обратите внимание, что `Dir$` поддерживает wild-cards, поэтому для проверки *определенного* файла переданное имя `pathName` должно быть проверено, чтобы убедиться, что оно не содержит их. В приведенном ниже примере возникает ошибка - если это не желаемое поведение, функцию можно изменить, чтобы просто вернуть `False` .

```
Public Function FileExists(pathName As String) As Boolean
    If InStr(1, pathName, "*") Or InStr(1, pathName, "?") Then
        'Exit Function   'Return False on wild-cards.
        Err.Raise 52    'Raise error on wild-cards.
    End If
    FileExists = Dir$(pathName) <> vbNullString
End Function
```

Папки (метод Dir $):

Функция `Dir$()` также может использоваться, чтобы определить, существует ли папка, указывая передачу `vbDirectory` для параметра необязательных `attributes` . В этом случае переданное значение `pathName` должно заканчиваться разделителем пути (`\`), поскольку совпадающие *имена файлов* будут вызывать ложные срабатывания. Имейте в виду, что wild-cards разрешены только после последнего разделителя путей, поэтому приведенная ниже функция примера будет вызывать ошибку времени выполнения 52 - «Плохое имя или номер файла», если вход содержит wild-card. Если это не является желаемым поведением, раскомментируйте `On Error Resume Next` в верхней части функции. Также помните, что `Dir$` поддерживает относительные пути к файлу (то есть `..\Foo\Bar`), поэтому результаты гарантируются только в том случае, если текущий рабочий каталог не изменяется.

```
Public Function FolderExists(ByVal pathName As String) As Boolean
    'Uncomment the "On Error" line if paths with wild-cards should return False
    'instead of raising an error.
    'On Error Resume Next
    If pathName = vbNullString Or Right$(pathName, 1) <> "\" Then
        Exit Function
    End If
    FolderExists = Dir$(pathName, vbDirectory) <> vbNullString
End Function
```

Папки (метод ChDir):

Оператор `ChDir` также может использоваться для проверки наличия папки. Обратите внимание, что этот метод временно изменит среду, в которой работает VBA, поэтому, если это необходимо, вместо этого следует использовать метод `Dir$` . У этого есть преимущество, заключающееся в том, что он намного менее прощает свой параметр. Этот метод также поддерживает относительные пути к файлам, поэтому имеет ту же оговорку, что и метод `Dir$` .

```
Public Function FolderExists(ByVal pathName As String) As Boolean
    'Cache the current working directory
    Dim cached As String
    cached = CurDir$

    On Error Resume Next
    ChDir pathName
    FolderExists = Err.Number = 0
    On Error GoTo 0
    'Change back to the cached working directory.
    ChDir cached
End Function
```

Создание и удаление файловых папок

ПРИМЕЧАНИЕ. Для краткости в приведенных ниже примерах используйте функцию FolderExists из примера « **Определение файлов и файлов»** в этом разделе.

Оператор `MkDir` может использоваться для создания новой папки. Он принимает пути, содержащие буквы дисков (`C:\Foo`), имена UNC (`\\Server\Foo`), относительные пути (`..\Foo`) или текущий рабочий каталог (`Foo`).

Если имя диска или UNC опущено (например, `\Foo`), папка создается на текущем диске. Это может быть или не быть тем же диском, что и текущий рабочий каталог.

```
Public Sub MakeNewDirectory(ByVal pathName As String)
    'MkDir will fail if the directory already exists.
    If FolderExists(pathName) Then Exit Sub
    'This may still fail due to permissions, etc.
    MkDir pathName
```

```
End Sub
```

Оператор `RmDir` может использоваться для удаления существующих папок. Он принимает пути в тех же формах, что и `MkDir` и использует то же отношение к текущему рабочему каталогу и диску. Обратите внимание, что оператор похож на команду оболочки Windows `rd` , поэтому будет вызывать ошибку времени выполнения 75: «Ошибка доступа к пути / файлу», если целевой каталог не пуст.

```
Public Sub DeleteDirectory(ByVal pathName As String)
    If Right$(pathName, 1) <> "\" Then
        pathName = pathName & "\"
    End If
    'Rmdir will fail if the directory doesn't exist.
    If Not FolderExists(pathName) Then Exit Sub
    'Rmdir will fail if the directory contains files.
    If Dir$(pathName & "*") <> vbNullString Then Exit Sub

    'Rmdir will fail if the directory contains directories.
    Dim subDir As String
    subDir = Dir$(pathName & "*", vbDirectory)
    Do
        If subDir <> "." And subDir <> ".." Then Exit Sub
        subDir = Dir$(, vbDirectory)
    Loop While subDir <> vbNullString

    'This may still fail due to permissions, etc.
    RmDir pathName
End Sub
```

Прочитайте Работа с файлами и каталогами без использования FileSystemObject онлайн: https://riptutorial.com/ru/vba/topic/5706/работа-с-файлами-и-каталогами-без-использования-filesystemobject

глава 34: Рекурсия

Вступление

Функция, которая вызывает себя, называется *рекурсивной* . Рекурсивная логика часто может быть реализована как цикл. Рекурсия должна контролироваться параметром, так что функция знает, когда прекратить рекурсию и углубить стек вызовов. *Бесконечная рекурсия в* конечном итоге вызывает ошибку времени выполнения «28»: «Вне пространства стека».

См. Рекурсия .

замечания

Рекурсия позволяет повторять вызовы с саморегуляцией процедуры.

Examples

Факториалы

```
Function Factorial(Value As Long) As Long
    If Value = 0 Or Value = 1 Then
         Factorial = 1
    Else
       Factorial = Factorial(Value - 1) * Value
    End If
End Function
```

Рекурсия папки

Ранняя привязка (со ссылкой на `Microsoft Scripting Runtime`)

```
    Sub EnumerateFilesAndFolders( _
        FolderPath As String, _
        Optional MaxDepth As Long = -1, _
        Optional CurrentDepth As Long = 0, _
        Optional Indentation As Long = 2)

        Dim FSO As Scripting.FileSystemObject
        Set FSO = New Scripting.FileSystemObject

        'Check the folder exists
        If FSO.FolderExists(FolderPath) Then
            Dim fldr As Scripting.Folder
            Set fldr = FSO.GetFolder(FolderPath)

            'Output the starting directory path
```

```
            If CurrentDepth = 0 Then
              Debug.Print fldr.Path
            End If

            'Enumerate the subfolders
            Dim subFldr As Scripting.Folder
            For Each subFldr In fldr.SubFolders
                Debug.Print Space$((CurrentDepth + 1) * Indentation) & subFldr.Name
                If CurrentDepth < MaxDepth Or MaxDepth = -1 Then
                    'Recursively call EnumerateFilesAndFolders
                    EnumerateFilesAndFolders subFldr.Path, MaxDepth, CurrentDepth + 1,
Indentation
                End If
            Next subFldr

            'Enumerate the files
            Dim fil As Scripting.File
            For Each fil In fldr.Files
                Debug.Print Space$((CurrentDepth + 1) * Indentation) & fil.Name
            Next fil
        End If
    End Sub
```

Прочитайте Рекурсия онлайн: https://riptutorial.com/ru/vba/topic/3236/рекурсия

глава 35: События

Синтаксис

- **Исходный модуль** : `[Public] Event [identifier]([argument_list])`
- **Модуль обработчика** : `Dim|Private|Public WithEvents [identifier] As [type]`

замечания

- Событие может быть только `Public` . Модификатор является необязательным, поскольку члены модуля модуля (включая события) по умолчанию неявно `Public` .
- Переменная `WithEvents` может быть `Private` или `Public` , но не `Friend` . Модификатор является обязательным, потому что `WithEvents` не является ключевым словом, которое объявляет переменную, но является частью ключевого слова модификатора синтаксиса объявления переменной. Следовательно, ключевое слово `Dim` должно использоваться, если модификатор доступа отсутствует.

Examples

Источники и обработчики

Что такое события?

VBA *управляется событиями* : код VBA запускается в ответ на события, вызванные хост-приложением или хост-документом - понимание событий является основополагающим для понимания VBA.

API часто выставляют объекты, которые вызывают ряд *событий* в ответ на различные состояния. Например, объект `Excel.Application` вызывает событие, когда новая рабочая книга создается, открывается, активируется или закрывается. Или когда рассчитывается рабочий лист. Или непосредственно перед сохранением файла. Или сразу после. Кнопка в форме вызывает событие `Click` когда пользователь нажимает на нее, сама пользовательская форма вызывает событие сразу после его активации, а другая перед закрытием.

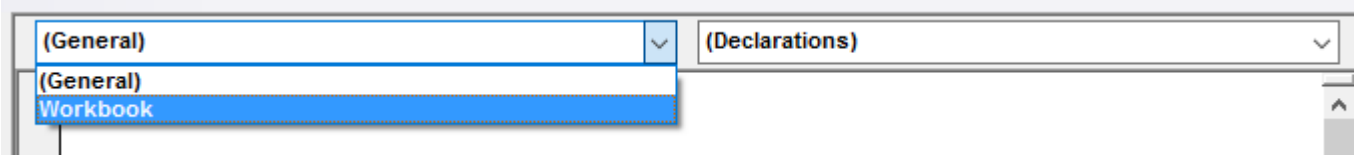
С точки зрения API, события являются *точками расширения* : клиентский код может выбрать реализацию кода, который *обрабатывает* эти события, и выполнять собственный код при каждом запуске этих событий: таким образом вы можете автоматически выполнять свой пользовательский код каждый раз, когда выбор изменяется на любом листе - путем

обработки события, которое запускается, когда выбор изменяется на любом рабочем листе.

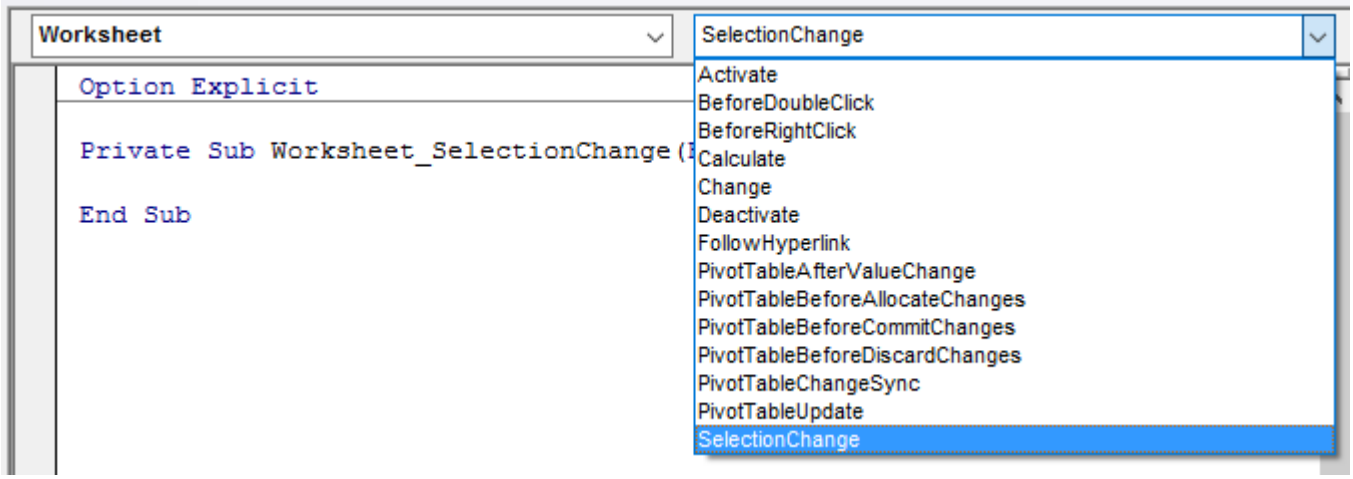
Объект, который предоставляет события, является источником *события* . Метод, обрабатывающий событие, является *обработчиком* .

Обработчики

Модули документов VBA (например, `ThisDocument` , `ThisWorkbook` , `Sheet1` и т. Д.) И модули `UserForm` являются *модулями классов,* которые *реализуют* специальные интерфейсы, которые выставляют несколько *событий* . Вы можете просматривать эти интерфейсы в выпадающем списке слева вверху панели кода:



В правом выпадающем списке перечислены элементы интерфейса, выбранные в раскрывающемся списке слева:



VBE автоматически генерирует заглушку обработчика событий, когда элемент выбран в правом списке, или перемещается туда, если обработчик существует.

Вы можете определить переменную `WithEvents` с модулем в любом модуле:

```
Private WithEvents Foo As Workbook
Private WithEvents Bar As Worksheet
```

Каждое объявление `WithEvents` становится доступным для выбора из раскрывающегося списка слева. Когда в правом выпадающем списке выбрано событие, VBE генерирует заглушку обработчика событий, названную после объекта `WithEvents` и имя события, соединенного с подчеркиванием:

```
Private WithEvents Foo As Workbook
Private WithEvents Bar As Worksheet

Private Sub Foo_Open()

End Sub

Private Sub Bar_SelectionChange(ByVal Target As Range)

End Sub
```

Только типы, которые выставляют хотя бы одно событие, могут использоваться с `WithEvents` , а объявлениям `WithEvents` нельзя назначить ссылку на месте с ключевым словом `New` . Этот код является незаконным:

```
Private WithEvents Foo As New Workbook 'illegal
```

Ссылка на объект должна быть `Set` явно; в модуле класса, хорошее место для этого часто `Class_Initialize` обработчике `Class_Initialize` , потому что тогда класс обрабатывает события этого объекта до тех пор, пока его экземпляр существует.

ИСТОЧНИКИ

Любой модуль класса (или модуль документа или пользовательская форма) может быть источником события. Используйте ключевое слово `Event` для определения *сигнатуры* для события, в разделе *объявлений* модуля:

```
Public Event SomethingHappened(ByVal something As String)
```

Подпись события определяет, как будет создаваться событие, и как будут выглядеть обработчики событий.

События могут быть *подняты* только внутри класса, в котором они определены: клиентский код может *обрабатывать* их только. События поднимаются с `RaiseEvent` словом `RaiseEvent` ; аргументы события приведены в этот момент:

```
Public Sub DoSomething()
    RaiseEvent SomethingHappened("hello")
End Sub
```

Без кода, обрабатывающего событие `SomethingHappened` , выполнение процедуры `DoSomething` все равно вызовет событие, но ничего не произойдет. Предполагая, что источником события является приведенный выше код в классе с именем `Something` , этот код в `ThisWorkbook` будет показывать окно сообщения «привет» всякий раз, когда `test.DoSomething` :

```
Private WithEvents test As Something
```

```
Private Sub Workbook_Open()
    Set test = New Something
    test.DoSomething
End Sub

Private Sub test_SomethingHappened(ByVal bar As String)
'this procedure runs whenever 'test' raises the 'SomethingHappened' event
    MsgBox bar
End Sub
```

Передача данных обратно в источник события

Использование параметров, переданных по ссылке

Событие может определять параметр `ByRef` должен быть возвращен вызывающему абоненту:

```
Public Event BeforeSomething(ByRef cancel As Boolean)
Public Event AfterSomething()

Public Sub DoSomething()
    Dim cancel As Boolean
    RaiseEvent BeforeSomething(cancel)
    If cancel Then Exit Sub

    'todo: actually do something

    RaiseEvent AfterSomething
End Sub
```

Если в событии `BeforeSomething` есть обработчик, который устанавливает свой параметр `cancel` в значение `True` , тогда, когда выполнение возвращается из обработчика, `cancel` будет `True` а `AfterSomething` никогда не будет поднята.

```
Private WithEvents foo As Something

Private Sub foo_BeforeSomething(ByRef cancel As Boolean)
    cancel = MsgBox("Cancel?", vbYesNo) = vbYes
End Sub

Private Sub foo_AfterSomething()
    MsgBox "Didn't cancel!"
End Sub
```

Предполагая, что ссылка на объект `foo` присваивается где-то, когда запускается `foo.DoSomething` , окно сообщения запрашивает, следует ли отменить, а второе поле сообщения говорит «не отменять» только тогда, когда выбрано « `Нет` » .

Использование изменяемых объектов

Вы также можете передать копию `ByVal` объекта `ByVal` и позволить обработчикам изменять свойства этого объекта; вызывающий может затем прочитать измененные значения свойств и действовать соответственно.

```
'class module ReturnBoolean
Option Explicit
Private encapsulated As Boolean

Public Property Get ReturnValue() As Boolean
'Attribute ReturnValue.VB_UserMemId = 0
    ReturnValue = encapsulated
End Property

Public Property Let ReturnValue(ByVal value As Boolean)
    encapsulated = value
End Property
```

В сочетании с типом `Variant` это можно использовать для создания довольно неочевидных способов вернуть значение вызывающему:

```
Public Event SomeEvent(ByVal foo As Variant)

Public Sub DoSomething()
    Dim result As ReturnBoolean
    result = New ReturnBoolean

    RaiseEvent SomeEvent(result)

    If result Then ' If result.ReturnValue Then
        'handler changed the value to True
    Else
        'handler didn't modify the value
    End If
End Sub
```

Обработчик будет выглядеть так:

```
Private Sub source_SomeEvent(ByVal foo As Variant) 'foo is actually a ReturnBoolean object
    foo = True 'True is actually assigned to foo.ReturnValue, the class' default member
End Sub
```

Прочитайте События онлайн: https://riptutorial.com/ru/vba/topic/5278/события

глава 36: Соглашения об именах

Examples

Переменные имена

Переменные содержат данные. Назовите их после того, для чего они используются, а **не после их типа** или области данных, используя **существительное** . Если вы вынуждены *пронумеровать* переменные (например , `thing1, thing2, thing3`), а затем рассмотреть вопрос об использовании соответствующей структуры данных вместо (например , массива, в `Collection` , или `Dictionary`).

Имена переменных, которые представляют собой итеративный *набор* значений - например, массив, `Collection` , `Dictionary` или `Range` ячеек, должны быть множественными.

Некоторые общие соглашения об именах VBA:

Для переменных уровня процедуры :

`camelCase`

```
Public Sub ExampleNaming(ByVal inputValue As Long, ByRef inputVariable As Long)

    Dim procedureVariable As Long
    Dim someOtherVariable As String

End Sub
```

Для переменных уровня модуля:

`PascalCase`

```
Public GlobalVariable As Long
Private ModuleVariable As String
```

Для констант:

`SHOUTY_SNAKE_CASE` обычно используется для дифференциации констант от переменных:

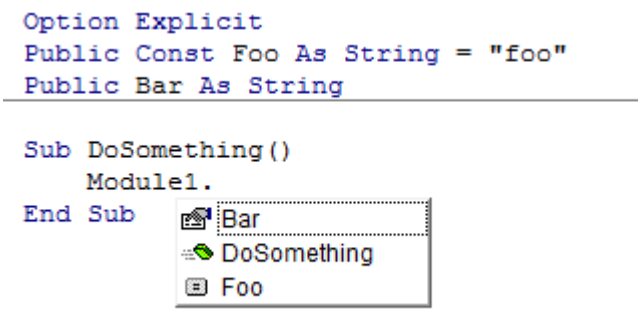
```
Public Const GLOBAL_CONSTANT As String = "Project Version #1.000.000.001"
Private Const MODULE_CONSTANT As String = "Something relevant to this Module"

Public Sub SomeProcedure()

    Const PROCEDURE_CONSTANT As Long = 10
```

```
End Sub
```

Однако имена `PascalCase` делают более чистый код и так же хороши, поскольку IntelliSense использует разные значки для переменных и констант:



Венгерская нотация

> Назовите их после того, для чего они используются, а **не после их типа** или области данных.

«Венгерская нотация облегчает просмотр типа переменной»

Если вы пишете свой код, например, процедуры придерживаетесь *принципа единой ответственности* (как и должно быть), вы никогда не должны смотреть на экранные объявления переменных в верхней части любой процедуры; объявлять переменные как можно ближе к их первому использованию, и их тип данных всегда будет на виду, если вы объявите их с явным типом. Ярлык `Ctrl` + `i` VBE можно использовать для отображения типа переменной в всплывающей подсказке.

Для какой переменной используется гораздо более полезная информация, чем ее тип данных, *особенно* на языке, таком как VBA, который счастливо и неявно преобразует тип в другой по мере необходимости.

Рассмотрим `iFile` и `strFile` в этом примере:

```
Function bReadFile(ByVal strFile As String, ByRef strData As String) As Boolean
    Dim bRetVal As Boolean
    Dim iFile As Integer

    On Error GoTo CleanFail

    iFile = FreeFile
    Open strFile For Input As #iFile
    Input #iFile, strData

    bRetVal = True

CleanExit:
```

```
    Close #iFile
    bReadFile = bRetVal
    Exit Function
CleanFail:
    bRetVal = False
    Resume CleanExit
End Function
```

Сравнить с:

```
Function CanReadFile(ByVal path As String, ByRef outContent As String) As Boolean
    On Error GoTo CleanFail

    Dim handle As Integer
    handle = FreeFile

    Open path For Input As #handle
    Input #handle, outContent

    Dim result As Boolean
    result = True

CleanExit:
    Close #handle
    CanReadFile = result
    Exit Function
CleanFail:
    result = False
    Resume CleanExit
End Function
```

`strData` передается `ByRef` в верхнем примере, но помимо того факта, что нам посчастливилось увидеть, что он *явно* передан как таковой, нет никаких указаний на то, что `strData` фактически *возвращается* функцией.

В нижнем примере называет его `outContent` ; это `out` префикса , что венгерская нотация была изобретена для: чтобы помочь прояснить , *что переменная используется для,* в данном случае четко определить его как «вне» параметра.

Это полезно, поскольку IntelliSense сам по себе не отображает `ByRef` , даже если параметр *явно* передан по ссылке:

```
Public Sub DoSomething()
    if CanReadFile(path, |
End Sub
```
CanReadFile(*ByVal path As String*, ***outContent As String***) As Boolean

Что приводит к...

Венгерский сделал правильно

Первоначально венгерская нотация не имела ничего общего с переменными типами . Фактически, венгерская нотация, *сделанная правильно* , действительно полезна. Рассмотрим этот небольшой пример (`ByVal` и `As Integer` удалены для краткости):

```
Public Sub Copy(iX1, iY1, iX2, iY2)
End Sub
```

Сравнить с:

```
Public Sub Copy(srcColumn, srcRow, dstColumn, dstRow)
End Sub
```

`src` и `dst` являются префиксами *венгерского нотации* здесь, и они передают *полезную* информацию, которая в противном случае уже не может быть выведена из имен параметров или IntelliSense, показывая нам объявленный тип.

Конечно, есть лучший способ передать все это, используя правильную *абстракцию* и реальные слова, которые могут быть произнесены вслух и понятны - как надуманный пример:

```
Type Coordinate
    RowIndex As Long
    ColumnIndex As Long
End Type

Sub Copy(source As Coordinate, destination As Coordinate)
End Sub
```

Имена процедур

Процедуры *что-то делают* . Назовите их после того, что они делают, используя **глагол** . Если точное написание процедуры невозможно, вероятно, процедура *выполняет слишком много вещей* и ее необходимо разбить на более мелкие, более специализированные процедуры.

Некоторые общие соглашения об именах VBA:

Для всех процедур:

`PascalCase`

```
Public Sub DoThing()

End Sub

Private Function ReturnSomeValue() As [DataType]

End Function
```

Для процедур обработчика событий:

`ObjectName_EventName`

```
Public Sub Workbook_Open()

End Sub

Public Sub Button1_Click()

End Sub
```

Обработчики событий обычно автоматически именуются VBE; переименование их без переименования объекта и / или обработанного события приведет к поломке кода - код будет запущен и скомпилирован, но процедура обработчика будет потеряна и никогда не будет выполнена.

Boolean Members

Рассмотрим функцию с булевым возвратом:

```
Function bReadFile(ByVal strFile As String, ByRef strData As String) As Boolean
End Function
```

Сравнить с:

```
Function CanReadFile(ByVal path As String, ByRef outContent As String) As Boolean
End Function
```

`Can` префикс *не* служат той же цели, что и `b` приставкой: он определяет возвращаемое значение функции как `Boolean` . Но `Can` читать лучше, чем `b` :

```
If CanReadFile(path, content) Then
```

По сравнению с:

```
If bReadFile(strFile, strData) Then
```

Рассмотрите возможность использования префиксов , например, `Can` , `Is` или `Has` перед Логическое возвращающих членов (функции и свойства), но только тогда , когда он добавляет ценность. Это соответствует действующим правилам именования Microsoft .

Прочитайте Соглашения об именах онлайн: https://riptutorial.com/ru/vba/topic/1184/соглашения-об-именах

глава 37: Создание пользовательского класса

замечания

В этой статье будет показано, как создать полный пользовательский класс в VBA. Он использует пример объекта `DateRange` , потому что начальная и конечная дата часто передаются вместе с функциями.

Examples

Добавление свойства в класс

Процедура `Property` представляет собой серию операторов, которые извлекают или изменяют настраиваемое свойство в модуле.

Существует три типа аксессуаров свойств:

1. A `Get` процедуру, возвращающую значение свойства.
2. A `Let` процедура, которая присваивает `Object` значение (non `Object`).
3. Процедура `Set` , которая назначает ссылку `Object` .

Атрибуты доступа к ресурсам часто определяются парами, используя как `Get` и `Let` / `Set` для каждого свойства. Свойство только с процедурой `Get` будет доступно только для чтения, а свойство с процедурой `Let` / `Set` будет только для записи.

В следующем примере для класса `DateRange` определены четыре свойства accessors:

1. `StartDate` (*чтение / запись*). Значение даты, представляющее более раннюю дату в диапазоне. Каждая процедура использует значение переменной модуля `mStartDate` .
2. `EndDate` (*чтение / запись*). Значение даты, представляющее более позднюю дату в диапазоне. Каждая процедура использует значение переменной модуля `mEndDate` .
3. `DaysBetween` (*только для чтения*). Вычисленное целочисленное значение, представляющее количество дней между двумя датами. Поскольку существует только процедура `Get` , это свойство не может быть изменено напрямую.
4. `RangeToCopy` (*только для записи*). Процедура `Set` , используемая для копирования значений существующего объекта `DateRange` .

```
Private mStartDate As Date                ' Module variable to hold the starting date
Private mEndDate As Date                  ' Module variable to hold the ending date

' Return the current value of the starting date
Public Property Get StartDate() As Date
```

```
    StartDate = mStartDate
End Property

' Set the starting date value. Note that two methods have the name StartDate
Public Property Let StartDate(ByVal NewValue As Date)
    mStartDate = NewValue
End Property

' Same thing, but for the ending date
Public Property Get EndDate() As Date
    EndDate = mEndDate
End Property

Public Property Let EndDate(ByVal NewValue As Date)
    mEndDate = NewValue
End Property

' Read-only property that returns the number of days between the two dates
Public Property Get DaysBetween() As Integer
    DaysBetween = DateDiff("d", mStartDate, mEndDate)
End Function

' Write-only property that passes an object reference of a range to clone
Public Property Set RangeToCopy(ByRef ExistingRange As DateRange)

Me.StartDate = ExistingRange.StartDate
Me.EndDate = ExistingRange.EndDate

End Property
```

Добавление функциональности в класс

Любые общедоступные `Sub` , `Function` или `Property` внутри модуля класса можно вызывать, предшествуя вызову с помощью ссылки на объект:

```
Object.Procedure
```

В классе `DateRange Sub` может использоваться для добавления количества дней до даты окончания:

```
Public Sub AddDays(ByVal NoDays As Integer)
    mEndDate = mEndDate + NoDays
End Sub
```

`Function` может вернуть последний день следующего месяца (обратите внимание, что `GetFirstDayOfMonth` не будет виден вне класса, поскольку он является закрытым):

```
Public Function GetNextMonthEndDate() As Date
    GetNextMonthEndDate = DateAdd("m", 1, GetFirstDayOfMonth())
End Function

Private Function GetFirstDayOfMonth() As Date
    GetFirstDayOfMonth = DateAdd("d", -DatePart("d", mEndDate), mEndDate)
End Function
```

Процедуры могут принимать аргументы любого типа, включая ссылки на объекты определяемого класса.

В следующем примере проверяется, имеет ли текущий объект `DateRange` дату начала и дату окончания, которая включает дату начала и окончания другого объекта `DateRange` .

```
Public Function ContainsRange(ByRef TheRange As DateRange) As Boolean
    ContainsRange = TheRange.StartDate >= Me.StartDate And TheRange.EndDate <= Me.EndDate
End Function
```

Обратите внимание на использование нотации `Me` как способ доступа к значению объекта, выполняющего код.

Класс модуля, возможности и повторное использование

По умолчанию новый класс-класс является частным, поэтому он доступен *только* для экземпляра и использования в VBProject, в котором он определен. Вы можете объявлять, создавать экземпляры и использовать класс в любом месте *одного и* того *же* проекта:

```
'Class List has Instancing set to Private
'In any other module in the SAME project, you can use:

Dim items As List
Set items = New List
```

Но часто вы будете писать классы, которые вы хотели бы использовать в других проектах, *не* копируя модуль между проектами. Если вы определяете класс `List` в `ProjectA` и хотите использовать этот класс в `ProjectB` , вам необходимо выполнить 4 действия:

1. Измените свойство `ProjectA` класса `List` в `ProjectA` в окне свойств, от `Private` до `PublicNotCreatable`
2. Создайте публичную «фабричную» функцию в `ProjectA` которая создает и возвращает экземпляр класса `List` . Обычно заводская функция включает аргументы для инициализации экземпляра класса. Функция фабрики требуется, потому что класс может использоваться `ProjectB` но `ProjectB` не может напрямую создать экземпляр класса `ProjectA` .

   ```
   Public Function CreateList(ParamArray values() As Variant) As List
       Dim tempList As List
       Dim itemCounter As Long
       Set tempList = New List
       For itemCounter = LBound(values) to UBound(values)
           tempList.Add values(itemCounter)
       Next itemCounter
       Set CreateList = tempList
   End Function
   ```

3. В `ProjectB` добавьте ссылку на `ProjectA` используя меню `Tools..References...`

4. В `ProjectB` объявите переменную и присвойте ей экземпляр `List` используя заводскую функцию из `ProjectA`

```
Dim items As ProjectA.List
Set items = ProjectA.CreateList("foo","bar")

'Use the items list methods and properties
items.Add "fizz"
Debug.Print items.ToString()
'Destroy the items object
Set items = Nothing
```

Прочитайте Создание пользовательского класса онлайн:
https://riptutorial.com/ru/vba/topic/4464/создание-пользовательского-класса

глава 38: Создание процедуры

Examples

Введение в процедуры

`Sub` - это процедура, которая выполняет определенную задачу, но не возвращает определенное значение.

```
Sub ProcedureName ([argument_list])
    [statements]
End Sub
```

Если не указан модификатор доступа, процедура является `Public` по умолчанию.

`Function` - это процедура, которая задает данные и возвращает значение, в идеале без глобальных или объемных побочных эффектов.

```
Function ProcedureName ([argument_list]) [As ReturnType]
    [statements]
End Function
```

`Property` - это процедура, которая *инкапсулирует* данные модуля. Свойство может иметь до 3-х аксессуаров: `Get` чтобы вернуть значение или ссылку на объект, `Let` назначить значение и / или `Set` для назначения ссылки на объект.

```
Property Get|Let|Set PropertyName([argument_list]) [As ReturnType]
    [statements]
End Property
```

Свойства обычно используются в модулях классов (хотя они также разрешены в стандартных модулях), предоставляя доступ к данным, которые в противном случае недоступны для вызывающего кода. Свойство, которое предоставляет только доступ к `Get` является «только для чтения»; свойство, которое будет раскрывать только атрибут `Let` и / или `Set` является «только для записи». Свойства Write-only не считаются хорошей практикой программирования - если код клиента может *написать* значение, он должен иметь возможность *прочитать* его обратно. Рассмотрите возможность выполнения процедуры `Sub` вместо создания свойства для записи.

Возврат значения

Процедура `Function` или `Property Get` может (и должна!) Возвращать значение своему вызывающему. Это делается путем назначения идентификатора процедуры:

```
Property Get Foo() As Integer
    Foo = 42
End Property
```

Функция с примерами

Как указано выше, функции представляют собой более мелкие процедуры, содержащие небольшие фрагменты кода, которые могут повторяться внутри процедуры.

Функции используются для уменьшения избыточности кода.

Подобно процедуре, функция может быть объявлена с или без списка аргументов.

Функция объявляется как возвращаемый тип, так как все функции возвращают значение. Имя и возвращаемая переменная функции являются одинаковыми.

1. Функция с параметром:

```
Function check_even(i as integer) as boolean
if (i mod  2) = 0 then
check_even = True
else
check_even=False
end if
end Function
```

2. Функция без параметра:

```
Function greet() as String
greet= "Hello Coder!"
end Function
```

Функция может быть вызвана различными способами внутри функции. Поскольку функция, объявленная с типом возвращаемого значения, в основном является переменной. он используется аналогично переменной.

Функциональные вызовы:

```
call greet() 'Similar to a Procedural call just allows the Procedure to use the
             'variable greet
string_1=greet() 'The Return value of the function is used for variable
                 'assignment
```

Кроме того, эту функцию можно также использовать в качестве условий для if и других условных операторов.

```
  for i = 1 to 10
  if check_even(i) then
  msgbox i & " is Even"
  else
```

```
    msgbox i & " is Odd"
    end if
    next i
```

Далее Функции могут иметь модификаторы, такие как By ref и By val для своих аргументов.

Прочитайте Создание процедуры онлайн: https://riptutorial.com/ru/vba/topic/1474/создание-процедуры

глава 39: Сортировка

Вступление

В отличие от платформы .NET, библиотека Visual Basic для приложений не включает процедуры сортировки массивов.

Существует два типа обходных путей: 1) реализация алгоритма сортировки с нуля или 2) использование подпрограмм сортировки в других общедоступных библиотеках.

Examples

Реализация алгоритма - Быстрая сортировка по одномерному массиву

Из функции сортировки массива VBA?

```
Public Sub QuickSort(vArray As Variant, inLow As Long, inHi As Long)

  Dim pivot   As Variant
  Dim tmpSwap As Variant
  Dim tmpLow  As Long
  Dim tmpHi   As Long

  tmpLow = inLow
  tmpHi = inHi

  pivot = vArray((inLow + inHi) \ 2)

  While (tmpLow <= tmpHi)

     While (vArray(tmpLow) < pivot And tmpLow < inHi)
        tmpLow = tmpLow + 1
     Wend

     While (pivot < vArray(tmpHi) And tmpHi > inLow)
        tmpHi = tmpHi - 1
     Wend

     If (tmpLow <= tmpHi) Then
        tmpSwap = vArray(tmpLow)
        vArray(tmpLow) = vArray(tmpHi)
        vArray(tmpHi) = tmpSwap
        tmpLow = tmpLow + 1
        tmpHi = tmpHi - 1
     End If

  Wend

  If (inLow < tmpHi) Then QuickSort vArray, inLow, tmpHi
  If (tmpLow < inHi) Then QuickSort vArray, tmpLow, inHi

End Sub
```

Использование библиотеки Excel для сортировки одномерного массива

Этот код использует класс `Sort` в библиотеке объектов Microsoft Excel.

Для дальнейшего ознакомления см .:

- Скопировать диапазон в виртуальный диапазон
- Как скопировать выбранный диапазон в заданный массив?

```
Sub testExcelSort()

Dim arr As Variant

InitArray arr
ExcelSort arr

End Sub

Private Sub InitArray(arr As Variant)

Const size = 10
ReDim arr(size)

Dim i As Integer

' Add descending numbers to the array to start
For i = 0 To size
    arr(i) = size - i
Next i

End Sub

Private Sub ExcelSort(arr As Variant)

' Ininitialize the Excel objects (required)
Dim xl As New Excel.Application
Dim wbk As Workbook
Set wbk = xl.Workbooks.Add
Dim sht As Worksheet
Set sht = wbk.ActiveSheet

' Copy the array to the Range object
Dim rng As Range
Set rng = sht.Range("A1")
Set rng = rng.Resize(UBound(arr, 1), 1)
rng.Value = xl.WorksheetFunction.Transpose(arr)

' Run the worksheet's sort routine on the Range
Dim MySort As Sort
Set MySort = sht.Sort

With MySort
    .SortFields.Clear
    .SortFields.Add rng, xlSortOnValues, xlAscending, xlSortNormal
    .SetRange rng
    .Header = xlNo
    .Apply
```

```
End With

' Copy the results back to the array
CopyRangeToArray rng, arr

' Clear the objects
Set rng = Nothing
wbk.Close False
xl.Quit

End Sub

Private Sub CopyRangeToArray(rng As Range, arr)

Dim i As Long
Dim c As Range

' Can't just set the array to Range.value (adds a dimension)
For Each c In rng.Cells
    arr(i) = c.Value
    i = i + 1
Next c

End Sub
```

Прочитайте Сортировка онлайн: https://riptutorial.com/ru/vba/topic/8836/сортировка

глава 40: Строковые литералы - экранирование, непечатаемые символы и линейные продолжения

замечания

Назначение строковых литералов в VBA ограничено ограничениями IDE и кодовой страницы настроек языка текущего пользователя. В приведенных выше примерах показаны особые случаи экранированных строк, специальных, непечатаемых строк и длинных строковых литералов.

При назначении строковых литералов, содержащих символы, специфичные для определенной кодовой страницы, вам может потребоваться рассмотреть проблемы интернационализации, назначив строку из отдельного файла ресурсов юникода.

Examples

Выход из символа

Синтаксис VBA требует, чтобы строковый литерал отображался внутри `"` меток», поэтому, когда ваша строка должна *содержать* кавычки, вам нужно будет избежать / добавить `"` символ с дополнительным `"` чтобы VBA понимал, что вы намерены `""` интерпретируется как `"` строка».

```
'The following 2 lines produce the same output
Debug.Print "The man said, ""Never use air-quotes"""
Debug.Print "The man said, " & """ & "Never use air-quotes" & """

'Output:
'The man said, "Never use air-quotes"
'The man said, "Never use air-quotes"
```

Присвоение длинных строковых литералов

Редактор VBA разрешает только 1023 символа в строке, но обычно только первые 100-150 символов видны без прокрутки. Если вам нужно назначить длинные строковые литералы, но вы хотите, чтобы ваш код читался, вам нужно будет использовать продолжение строки и конкатенацию для назначения вашей строки.

```
Debug.Print "Lorem ipsum dolor sit amet, consectetur adipiscing elit. " & _
            "Integer hendrerit maximus arcu, ut elementum odio varius " & _
            "nec. Integer ipsum enim, iaculis et egestas ac, condiment" & _
```

```
            "um ut tellus."
'Output:
'Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer hendrerit maximus arcu, ut
elementum odio varius nec. Integer ipsum enim, iaculis et egestas ac, condimentum ut tellus.
```

VBA позволит вам использовать ограниченное количество строк (фактическое число зависит от длины каждой строки в непрерывном блоке), поэтому, если у вас очень длинные строки, вам необходимо назначить и повторно назначить с помощью конкатенации ,

```
Dim loremIpsum As String

'Assign the first part of the string
loremIpsum = "Lorem ipsum dolor sit amet, consectetur adipiscing elit. " & _
              "Integer hendrerit maximus arcu, ut elementum odio varius "
'Re-assign with the previous value AND the next section of the string
loremIpsum = loremIpsum & _
             "nec. Integer ipsum enim, iaculis et egestas ac, condiment" & _
             "um ut tellus."

Debug.Print loremIpsum

'Output:
'Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer hendrerit maximus arcu, ut
elementum odio varius nec. Integer ipsum enim, iaculis et egestas ac, condimentum ut tellus.
```

Использование строковых констант VBA

VBA определяет ряд строковых констант для специальных символов, таких как:

- vbCr: Carriage-Return 'То же, что и "\ r" в языках стиля C.
- vbLf: Line-Feed 'То же, что и "\ n" в языках стиля C.
- vbCrLf: Carriage-Return & Line-Feed (новая строка в Windows)
- vbTab: символ табуляции
- vbNullString: пустая строка, например ""

Вы можете использовать эти константы с конкатенацией и другими строковыми функциями для создания строковых литералов со специальными символами.

```
Debug.Print "Hello " & vbCrLf & "World"
'Output:
'Hello
'World

Debug.Print vbTab & "Hello" & vbTab & "World"
'Output:
'    Hello    World

Dim EmptyString As String
EmptyString = vbNullString
Debug.Print EmptyString = ""
'Output:
'True
```

Использование `vbNullString` считается лучшей практикой, чем эквивалентное значение `""` из-за различий в том, как компилируется код. Доступ к `vbNullString` осуществляется с помощью указателя на выделенную область памяти, а компилятор VBA достаточно умен, чтобы использовать нулевой указатель для представления `vbNullString` . Литерал `""` выделяется память, как если бы это был вариант с типизированным строком, что делает использование константы намного более эффективным:

```
Debug.Print StrPtr(vbNullString)    'Prints 0.
Debug.Print StrPtr("")              'Prints a memory address.
```

Прочитайте Строковые литералы - экранирование, непечатаемые символы и линейные продолжения онлайн: https://riptutorial.com/ru/vba/topic/3445/строковые-литералы---экранирование--непечатаемые-символы-и-линейные-продолжения

глава 41: Структуры данных

Вступление

[TODO: Эта тема должна быть примером всех базовых структур данных CS 101, а также некоторое объяснение как обзор того, как структуры данных могут быть реализованы в VBA. Это было бы хорошей возможностью связать и укрепить концепции, включенные в связанные с классом темы в документации VBA.]

Examples

Связанный список

Этот пример связанного списка реализует операции набора абстрактных данных .

Класс **SinglyLinkedNode**

```
Option Explicit

Private Value As Variant
Private NextNode As SinglyLinkedNode '"Next" is a keyword in VBA and therefore is not a valid
variable name
```

Класс **LinkedList**

```
Option Explicit

Private head As SinglyLinkedNode

'Set type operations

Public Sub Add(value As Variant)
    Dim node As SinglyLinkedNode

    Set node = New SinglyLinkedNode
    node.value = value
    Set node.nextNode = head

    Set head = node
End Sub

Public Sub Remove(value As Variant)
    Dim node As SinglyLinkedNode
    Dim prev As SinglyLinkedNode

    Set node = head

    While Not node Is Nothing
        If node.value = value Then
            'remove node
```

```
            If node Is head Then
                Set head = node.nextNode
            Else
                Set prev.nextNode = node.nextNode
            End If
            Exit Sub
        End If
        Set prev = node
        Set node = node.nextNode
    Wend

End Sub

Public Function Exists(value As Variant) As Boolean
    Dim node As SinglyLinkedNode

    Set node = head
    While Not node Is Nothing
        If node.value = value Then
            Exists = True
            Exit Function
        End If
        Set node = node.nextNode
    Wend
End Function

Public Function Count() As Long
    Dim node As SinglyLinkedNode

    Set node = head

    While Not node Is Nothing
        Count = Count + 1
        Set node = node.nextNode
    Wend

End Function
```

Двоичное дерево

Это пример несбалансированного дерева двоичного поиска . Бинарное дерево структурировано концептуально как иерархия узлов, нисходящих вниз от общего корня, где каждый узел имеет двух детей: слева и справа. Например, предположим, что числа 7, 5, 9, 3, 11, 6, 12, 14 и 15 были вставлены в BinaryTree. Структура будет такой, как показано ниже. Обратите внимание, что это двоичное дерево не сбалансировано , что может быть желательным признаком гарантирования производительности поисковых запросов - см. Деревья AVL на примере самобалансирующегося двоичного дерева поиска.

```
          7
         / \
        5   9
       / \   \
      3   6   11
                \
                 12
                   \
```

```
                      14
                       \
                        15
```

Класс **BinaryTreeNode**

```
Option Explicit

Public left As BinaryTreeNode
Public right As BinaryTreeNode
Public key As Variant
Public value As Variant
```

Класс **BinaryTree**

[СДЕЛАТЬ]

Прочитайте Структуры данных онлайн: https://riptutorial.com/ru/vba/topic/8628/структуры-данных

глава 42: Структуры контроля потока

Examples

Выберите Случай

`Select Case` можно использовать, когда возможны различные условия. Условия проверяются сверху вниз, и выполняется только первый случай, который соответствует.

```
Sub TestCase()
    Dim MyVar As String

    Select Case MyVar    'We Select the Variable MyVar to Work with
        Case "Hello"     'Now we simply check the cases we want to check
            MsgBox "This Case"
        Case "World"
            MsgBox "Important"
        Case "How"
            MsgBox "Stuff"
        Case "Are"
            MsgBox "I'm running out of ideas"
        Case "You?", "Today"  'You can separate several conditions with a comma
            MsgBox "Uuuhm..." 'if any is matched it will go into the case
        Case Else             'If none of the other cases is hit
            MsgBox "All of the other cases failed"
    End Select

    Dim i As Integer
    Select Case i
        Case Is > 2 '"Is" can be used instead of the variable in conditions.
            MsgBox "i is greater than 2"
        'Case 2 < Is '"Is" can only be used at the beginning of the condition.
        'Case Else is optional
    End Select
End Sub
```

Логика блока `Select Case` может быть инвертирована для поддержки тестирования различных переменных. В этом сценарии мы также можем использовать логические операторы:

```
Dim x As Integer
Dim y As Integer

x = 2
y = 5

Select Case True
    Case x > 3
        MsgBox "x is greater than 3"
    Case y < 2
        MsgBox "y is less than 2"
    Case x = 1
        MsgBox "x is equal to 1"
```

```
    Case x = 2 Xor y = 3
        MsgBox "Go read about ""Xor"""
    Case Not y = 5
        MsgBox "y is not 5"
    Case x = 3 Or x = 10
        MsgBox "x = 3 or 10"
    Case y < 10 And x < 10
        MsgBox "x and y are less than 10"
    Case Else
        MsgBox "No match found"
End Select
```

Операторы case также могут использовать арифметические операторы. Если для значения `Select Case` используется арифметический оператор, ему должно предшествовать ключевое слово `Is` :

```
Dim x As Integer

x = 5

Select Case x
    Case 1
        MsgBox "x equals 1"
    Case 2, 3, 4
        MsgBox "x is 2, 3 or 4"
    Case 7 To 10
        MsgBox "x is between 7 and 10 (inclusive)"
    Case Is < 2
        MsgBox "x is less than one"
    Case Is >= 7
        MsgBox "x is greater than or equal to 7"
    Case Else
        MsgBox "no match found"
End Select
```

Для каждого цикла

Конструкция `For Each` цикла идеально подходит для итерации всех элементов коллекции.

```
Public Sub IterateCollection(ByVal items As Collection)

    'For Each iterator must always be variant
    Dim element As Variant

    For Each element In items
        'assumes element can be converted to a string
        Debug.Print element
    Next

End Sub
```

Использовать `For Each` при повторной сборке объектов:

```
Dim sheet As Worksheet
For Each sheet In ActiveWorkbook.Worksheets
```

```
    Debug.Print sheet.Name
Next
```

Избегайте `For Each` при повторении массивов; `For` цикла будет предлагать значительно более высокую производительность с массивами. И наоборот, `For Each` цикла будет предлагать более высокую производительность при переборе в `Collection` .

Синтаксис

```
For Each [item] In [collection]
    [statements]
Next [item]
```

Ключевое слово `Next` может необязательно сопровождаться переменной итератора; это может помочь прояснить вложенные циклы, хотя есть лучшие способы прояснить вложенный код, например, извлечение внутреннего цикла в его собственную процедуру.

```
Dim book As Workbook
For Each book In Application.Workbooks

    Debug.Print book.FullName

    Dim sheet As Worksheet
    For Each sheet In ActiveWorkbook.Worksheets
        Debug.Print sheet.Name
    Next sheet
Next book
```

Проведите цикл

```
 Public Sub DoLoop()
    Dim entry As String
    entry = ""
    'Equivalent to a While loop will ask for strings until "Stop" in given
    'Prefer using a While loop instead of this form of Do loop
    Do While entry <> "Stop"
        entry = InputBox("Enter a string, Stop to end")
        Debug.Print entry
    Loop

    'Equivalent to the above loop, but the condition is only checked AFTER the
    'first iteration of the loop, so it will execute even at least once even
    'if entry is equal to "Stop" before entering the loop (like in this case)
    Do
        entry = InputBox("Enter a string, Stop to end")
        Debug.Print entry
    Loop While entry <> "Stop"


    'Equivalent to writing Do While Not entry="Stop"
    '
    'Because the Until is at the top of the loop, it will
```

```
    'not execute because entry is still equal to "Stop"
    'when evaluating the condition
    Do Until entry = "Stop"
        entry = InputBox("Enter a string, Stop to end")
        Debug.Print entry
    Loop

    'Equivalent to writing Do ... Loop While Not i >= 100
    Do
        entry = InputBox("Enter a string, Stop to end")
        Debug.Print entry
    Loop Until entry = "Stop"
End Sub
```

Пока цикл

```
'Will return whether an element is present in the array
Public Function IsInArray(values() As String, ByVal whatToFind As String) As Boolean
    Dim i As Integer
    i = 0

    While i < UBound(values) And values(i) <> whatToFind
        i = i + 1
    Wend

    IsInArray = values(i) = whatToFind
End Function
```

Для цикла

Цикл `For` используется для повторения замкнутого раздела кода заданное количество раз. Следующий простой пример иллюстрирует основной синтаксис:

```
Dim i as Integer           'Declaration of i
For i = 1 to 10            'Declare how many times the loop shall be executed
    Debug.Print i          'The piece of code which is repeated
Next i                     'The end of the loop
```

В приведенном выше коде объявляется целочисленный `i` . Цикл `For` присваивает каждому значению от 1 до 10 до `i` а затем выполняет `Debug.Print i` - то есть код печатает числа с 1 по 10 в ближайшее окно. Обратите внимание, что переменная цикла увеличивается с помощью оператора `Next` , то есть после выполнения прилагаемого кода, а не перед его выполнением.

По умолчанию счетчик будет увеличиваться на 1 при каждом выполнении цикла. Тем не менее, может быть указан `Step` , чтобы изменить величину приращения в виде литерального или возвращаемого значения функции. Если начальное значение, конечное значение или значение `Step` - это число с плавающей запятой, оно округляется до ближайшего целочисленного значения. `Step` может быть положительным или отрицательным.

```
Dim i As Integer
For i = 1 To 10 Step 2
    Debug.Print i       'Prints 1, 3, 5, 7, and 9
Next
```

В общем случае цикл `For` будет использоваться в ситуациях, когда это известно до того, как цикл запустит сколько раз для выполнения заключенного кода (в противном случае может быть более целесообразным цикл `Do` или `While`). Это связано с тем, что условие выхода фиксируется после первого входа в цикл, поскольку этот код демонстрирует:

```
Private Iterations As Long              'Module scope

Public Sub Example()
    Dim i As Long
    Iterations = 10
    For i = 1 To Iterations
        Debug.Print Iterations     'Prints 10 through 1, descending.
        Iterations = Iterations - 1
    Next
End Sub
```

Цикл `For` может быть выведен на раннем этапе с помощью инструкции `Exit For` :

```
Dim i As Integer

For i = 1 To 10
    If i > 5 Then
        Exit For
    End If
    Debug.Print i       'Prints 1, 2, 3, 4, 5 before loop exits early.
Next
```

Прочитайте Структуры контроля потока онлайн: https://riptutorial.com/ru/vba/topic/1873/структуры-контроля-потока

глава 43: Типы данных и ограничения

Examples

Байт

```
Dim Value As Byte
```

Байт представляет собой неподписанный 8-битный тип данных. Он может представлять целые числа от 0 до 255, и попытка сохранить значение за пределами этого диапазона приведет к ошибке времени выполнения 6: `Overflow` . Байт является единственным внутренним беззнаковым типом, доступным в VBA.

Функция кастинга для преобразования в байт - `CByte()` . Для бросков из типов с плавающей точкой результат округляется до ближайшего целочисленного значения с округлением .5.

Байт-массивы и строки

Строки и байтовые массивы могут быть заменены друг на друга посредством простого присваивания (необязательные функции преобразования).

Например:

```
Sub ByteToStringAndBack()

Dim str As String
str = "Hello, World!"

Dim byt() As Byte
byt = str

Debug.Print byt(0)   ' 72

Dim str2 As String
str2 = byt

Debug.Print str2     ' Hello, World!

End Sub
```

Чтобы иметь возможность кодировать символы Unicode , каждый символ в строке занимает два байта в массиве с первым младшим байтом. Например:

```
Sub UnicodeExample()

Dim str As String
str = ChrW(&H2123) & "."  ' Versicle character and a dot

Dim byt() As Byte
```

```
byt = str

Debug.Print byt(0), byt(1), byt(2), byt(3)   ' Prints: 35,33,46,0

End Sub
```

целое число

```
Dim Value As Integer
```

Целое число - это подписанный 16-битный тип данных. Он может хранить целые числа в диапазоне от -32,768 до 32,767, и попытка сохранить значение за пределами этого диапазона приведет к ошибке времени выполнения 6: переполнение.

Целые числа хранятся в памяти как малозначные значения с отрицаниями, представленными как дополнение к двум .

Обратите внимание, что в целом лучше использовать Long, а не Integer, если меньший тип не является членом типа или не требуется (либо по API-вызову, либо по другой причине), чтобы быть 2 байтами. В большинстве случаев VBA обрабатывает целые числа как 32-битные внутри, поэтому обычно нет преимуществ при использовании меньшего типа. Кроме того, каждый раз, когда используется тип Integer, он выполняет штраф за производительность, поскольку он бесшумно применяется как Long.

Функция кастинга для преобразования в Integer - это `CInt()` . Для бросков из типов с плавающей точкой результат округляется до ближайшего целочисленного значения с округлением .5.

логический

```
Dim Value As Boolean
```

Логическое значение используется для хранения значений, которые могут быть представлены как True или False. Внутренне тип данных сохраняется как 16-битное значение с 0, представляющим False, и любое другое значение, представляющее True.

Следует отметить, что когда Boolean применяется к числовому типу, все биты имеют значение 1. Это приводит к внутреннему представлению -1 для подписанных типов и максимальному значению для неподписанного типа (байт).

```
Dim Example As Boolean
Example = True
Debug.Print CInt(Example)  'Prints -1
Debug.Print CBool(42)      'Prints True
Debug.Print CByte(True)    'Prints 255
```

Функция кастинга для преобразования в булевское значение - `CBool()` . Несмотря на то, что

он представлен внутренне как 16-битное число, отбрасывание в Boolean из значений вне этого диапазона безопасно от переполнения, хотя оно устанавливает все 16 бит в 1:

```
Dim Example As Boolean
Example = CBool(2 ^ 17)
Debug.Print CInt(Example)   'Prints -1
Debug.Print CByte(Example)  'Prints 255
```

Долго

```
Dim Value As Long
```

A Long - это подписанный 32-битный тип данных. Он может хранить целые числа в диапазоне от -2,147,483,648 до 2,147,483,647, и попытка сохранить значение за пределами этого диапазона приведет к ошибке времени выполнения 6: переполнение.

Длинны хранятся в памяти как малозначные значения с отрицаниями, представленными как дополнение к двум .

Обратите внимание, что поскольку Long соответствует ширине указателя в 32-разрядной операционной системе, Longs обычно используются для хранения и передачи указателей на и из функций API.

Функция кастинга для преобразования в Long - `CLng()` . Для бросков из типов с плавающей точкой результат округляется до ближайшего целочисленного значения с округлением .5.

не замужем

```
Dim Value As Single
```

A Single - это подписанный 32-битный тип данных с плавающей точкой. Он хранится внутренне с использованием малогабаритной схемы памяти IEEE 754 . Таким образом, не существует фиксированного диапазона значений, которые могут быть представлены типом данных. Ограничение - это точность сохраненного значения. Единица может хранить значения ***целочисленных*** значений в диапазоне от -16,777,216 до 16,777,216 без потери точности. Точность чисел с плавающей запятой зависит от экспоненты.

Единица будет переполняться при назначении значения, превышающего примерно 2^{128} . Он не будет переполняться отрицательными показателями, хотя допустимая точность будет сомнительной до того, как будет достигнут верхний предел.

Как и во всех числах с плавающей запятой, следует соблюдать осторожность при проведении сравнений равенства. Лучшей практикой является включение значения дельта, соответствующего требуемой точности.

Функция кастинга для преобразования в Single - это `CSng()` .

двойной

```
Dim Value As Double
```

A Double - это подписанный 64-битный тип данных с плавающей точкой. Подобно Single , он хранится внутренне с использованием малогабаритной схемы памяти IEEE 754, и должны быть приняты те же меры предосторожности в отношении точности. Двойной объект может хранить ***целые*** значения в диапазоне от -9,007,199,254,740,992 до 9,007,199,254,740,992 без потери точности. Точность чисел с плавающей запятой зависит от экспоненты.

Двойной переполняется при назначении значения, превышающего примерно 2^{1024} . Он не будет переполняться отрицательными показателями, хотя допустимая точность будет сомнительной до того, как будет достигнут верхний предел.

Функция кастинга для преобразования в Double - это `CDbl()` .

валюта

```
Dim Value As Currency
```

Валюта - это подписанный 64-битный тип данных с плавающей точкой, аналогичный Double , но масштабированный на 10 000, чтобы обеспечить большую точность для 4 цифр справа от десятичной точки. Переменная Currency может хранить значения от -922,337,203,685,477.5808 до 922,337,203,685,477.5807, предоставляя ей наибольшую емкость любого встроенного типа в 32-битном приложении. Как следует из названия типа данных, считается, что наилучшим образом использовать этот тип данных при представлении денежных расчетов, поскольку масштабирование помогает избежать ошибок округления.

Функция кастинга для преобразования в валюту - `CCur()` .

Дата

```
Dim Value As Date
```

Тип Даты представлен внутренне в качестве подписанного 64 бит с плавающей точкой типа данных со значением слева от десятичного представляющего количества дней с момента эпохи декабря 30 - го, 1899 (хотя см примечания ниже). Значение справа от десятичного знака представляет время как дробный день. Таким образом, целочисленная дата будет иметь временной компонент в 12:00:00 AM, а x.5 будет иметь временной компонент в 12:00:00 PM.

Допустимые значения Даты находятся между 1 января 100 года и 31 декабря 9999. й

Поскольку двойной имеет больший диапазон, можно переполнять Дата путем присвоения значения вне этого диапазона.

Таким образом, его можно использовать взаимозаменяемо с расчетами Double for Date:

```
Dim MyDate As Double
MyDate = 0                                  'Epoch date.
Debug.Print Format$(MyDate, "yyyy-mm-dd")   'Prints 1899-12-30.
MyDate = MyDate + 365
Debug.Print Format$(MyDate, "yyyy-mm-dd")   'Prints 1900-12-30.
```

Функция кастинга для преобразования в дату - это `CDate()` , которая принимает любое числовое представление даты / времени с числовым типом. Важно отметить, что строковые представления дат будут преобразованы на основе текущей настройки локали, используемой, поэтому следует избегать прямых бросков, если код предназначен для переносимости.

строка

Строка представляет последовательность символов и поставляется в двух вариантах:

Переменная длина

```
Dim Value As String
```

Строка переменной длины позволяет добавлять и усекать и хранится в памяти как COM BSTR . Это состоит из 4-байтового беззнакового целого числа, которое хранит длину строки в байтах, за которой следуют строковые данные в виде широких символов (2 байта на символ) и заканчивается двумя нулевыми байтами. Таким образом, максимальная длина строки, которую может обрабатывать VBA, составляет 2 147 483 647 символов.

Внутренний указатель на структуру (восстанавливаемый `StrPtr()`) указывает на расположение памяти *данных* , а не на префикс длины. Это означает, что VBA String может быть передана непосредственно API-функциям, которым требуется указатель на массив символов.

Поскольку длина может изменяться, VBA перераспределяет память для String *каждый раз, когда назначается переменная* , которая может налагать штрафы за производительность для процедур, которые изменяют их повторно.

Фиксированная длина

```
Dim Value As String * 1024    'Declares a fixed length string of 1024 characters.
```

Строки фиксированной длины выделяются по 2 байта для каждого символа и сохраняются

в памяти как простой массив байтов. После выделения длина строки неизменна. Они не ***имеют*** нулевой конец в памяти, поэтому строка, заполняющая память, выделенную ненулевыми символами, непригодна для передачи функциям API, ожидающих нулевую завершаемую строку.

Строки фиксированной длины несут на себе устаревшее ограничение на 16 бит, поэтому длина может составлять до 65 535 символов. Попытка присвоить значение дольше, чем доступное пространство памяти, не приведет к ошибке выполнения - вместо этого результирующее значение будет просто усечено:

```
Dim Foobar As String * 5
Foobar = "Foo" & "bar"
Debug.Print Foobar          'Prints "Fooba"
```

Функция каста для преобразования в строку любого типа - `CStr()` .

Долго долго

```
Dim Value As LongLong
```

LongLong - это подписанный 64-битный тип данных и доступен только в 64-битных приложениях. Он **не** доступен в 32 - разрядных приложений , работающих на 64 - разрядных операционных системах. Он может хранить целые значения в диапазоне от -9,223,372,036,854,775,808 до 9,223,372,036,854,775,807, и попытка сохранить значение за пределами этого диапазона приведет к ошибке времени выполнения 6: переполнение.

LongLongs хранятся в памяти как малозначные значения с отрицаниями, представленными как дополнение к двум .

Тип данных LongLong был представлен как часть поддержки 64-битной операционной системы VBA. В 64-битных приложениях это значение может использоваться для хранения и передачи указателей на 64-битные API.

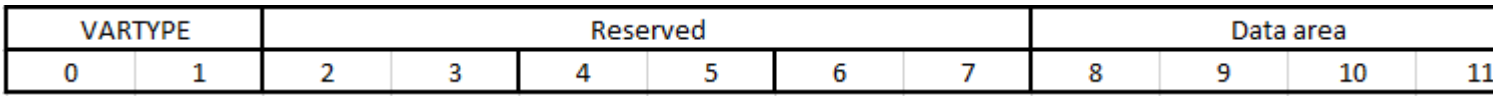
Функция литья для преобразования в LongLong - `CLngLng()` . Для бросков из типов с плавающей точкой результат округляется до ближайшего целочисленного значения с округлением .5.

Вариант

```
Dim Value As Variant    'Explicit
Dim Value               'Implicit
```

A Variant - это тип данных COM, который используется для хранения и обмена значениями произвольных типов, а любой другой тип в VBA может быть назначен Variant. Переменные, объявленные без явного типа, указанные `As [Type]` умолчанию для варианта.

Варианты хранятся в памяти как структура VARIANT, которая состоит из дескриптора байтового типа (VARTYPE), за которым следуют 6 зарезервированных байтов, а затем 8-байтовая область данных. Для числовых типов (включая Date и Boolean) базовое значение сохраняется в самом Variant. Для всех других типов область данных содержит указатель на базовое значение.



Основной тип Variant может быть определен с помощью функции `VarType()` которая возвращает числовое значение, хранящееся в дескрипторе типа, или функцию `TypeName()` которая возвращает строковое представление:

```
Dim Example As Variant
Example = 42
Debug.Print VarType(Example)    'Prints 2 (VT_I2)
Debug.Print TypeName(Example)   'Prints "Integer"
Example = "Some text"
Debug.Print VarType(Example)    'Prints 8 (VT_BSTR)
Debug.Print TypeName(Example)   'Prints "String"
```

Поскольку Variant может хранить значения любого типа, назначения из литералов без подсказок типа будут неявно отнесены к варианту соответствующего типа в соответствии с приведенной ниже таблицей. Литералы с типом намеков будут приведены к варианту намеченного типа.

Значение	Результирующий тип
Строковые значения	строка
Номера без плавающей запятой в диапазоне Integer	целое число
Номера без плавающей запятой в дальнем расстоянии	Долго
Номера без плавающей запятой вне дальнего расстояния	двойной
Все числа с плавающей запятой	двойной

Примечание. Если не существует конкретной причины использовать вариант (т.е. итератор в цикле For Each или требование API), тип обычно следует избегать для обычных задач по следующим причинам:

- Они не безопасны для типов, что увеличивает вероятность ошибок во время выполнения. Например, вариант, содержащий значение Integer, беззвучно изменяет себя на Long, а не на переполнение.
- Они вносят накладные расходы на обработку, требуя, по крайней мере, одного дополнительного разыменования указателя.

- Требование к памяти для Variant всегда ***на*** 8 байт выше, чем требуется для хранения базового типа.

Функция кастинга для преобразования в вариант - это `CVar()` .

LongPtr

```
Dim Value As LongPtr
```

LongPtr был представлен в VBA для поддержки 64-битных платформ. В 32-битной системе он рассматривается как Long и на 64-битных системах, он рассматривается как LongLong .

Основное использование заключается в предоставлении переносного способа хранения и передачи указателей на обе архитектуры (см. Изменение поведения кода во время компиляции .

Хотя он обрабатывается операционной системой как адрес памяти при использовании в вызовах API, следует отметить, что VBA рассматривает его как подписанный тип (и, следовательно, подвержен неподписанному сигналу переполнения). По этой причине любая арифметика указателя, выполняемая с использованием LongPtrs, не должна использовать `>` или `<` сравнения. Этот «quirk» также позволяет добавлять простые смещения, указывающие на действительные адреса в памяти, может привести к ошибкам переполнения, поэтому следует соблюдать осторожность при работе с указателями в VBA.

Функция кастинга для преобразования в LongPtr - `CLngPtr()` . Для бросков из типов с плавающей точкой результат округляется до ближайшего целочисленного значения с округлением .5 (хотя, поскольку он обычно является адресом памяти, использование его в качестве целевой цели для расчета с плавающей точкой в лучшем случае опасно).

Десятичный

```
Dim Value As Variant
Value = CDec(1.234)

'Set Value to the smallest possible Decimal value
Value = CDec("0.00000000000000000000000001")
```

Тип данных `Decimal` доступен *только* в качестве `CDec` `Variant` , поэтому вы должны объявить любую переменную, которая должна содержать `Decimal` как `Variant` а *затем* назначить `Decimal` значение с `CDec` функции `CDec` . Ключевое слово `Decimal` является зарезервированным словом (которое предполагает, что VBA в конечном итоге будет добавлять поддержку первого класса для этого типа), поэтому `Decimal` нельзя использовать как имя переменной или процедуры.

Для типа `Decimal` требуется 14 байт памяти (в дополнение к байтам, которые требуются родительскому варианту), и может хранить номера до 28 знаков после запятой. Для чисел

без знаков после запятой диапазон допустимых значений составляет -79,228,162,514,264,337,593,543,950,335 до +79,228,162,514,264,337,593,543,950,335 включительно. Для чисел с максимальным числом знаков в 28 знаков после запятой диапазон допустимых значений составляет -7.9228162514264337593543950335 по +7.9228162514264337593543950335 включительно.

Прочитайте Типы данных и ограничения онлайн: https://riptutorial.com/ru/vba/topic/3418/типы-данных-и-ограничения

глава 44: Условная компиляция

Examples

Изменение поведения кода во время компиляции

Директива `#Const` используется для определения пользовательской константы препроцессора. Они позже могут использоваться `#If` для контроля того, какие блоки кода компилируются и выполняются.

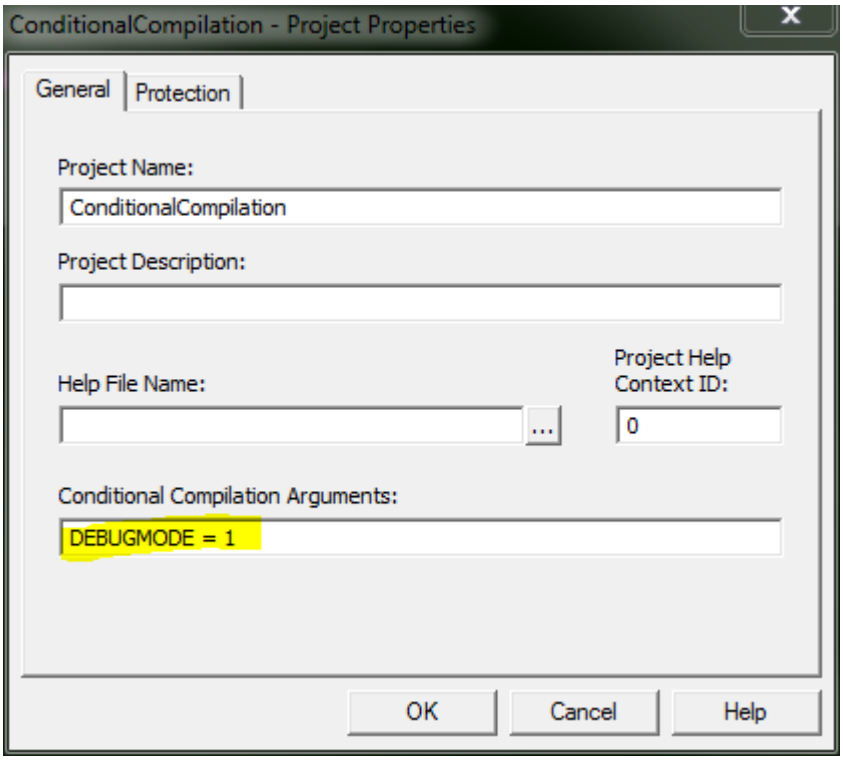
```
#Const DEBUGMODE = 1

#If DEBUGMODE Then
    Const filepath As String = "C:\Users\UserName\Path\To\File.txt"
#Else
    Const filepath As String = "\\server\share\path\to\file.txt"
#End If
```

Это приводит к тому, что значение параметра `filepath` устанавливается на `"C:\Users\UserName\Path\To\File.txt"` . Удаление `#Const` линии, или изменить его на `#Const DEBUGMODE = 0` приведет к `filepath` быть установлен на `"\\server\share\path\to\file.txt"` .

#Const Scope

Директива `#Const` действует только для одного файла кода (модуля или класса). Он должен быть объявлен для каждого файла, в который вы хотите использовать свою пользовательскую константу. Альтернативно, вы можете объявить `#Const` глобально для своего проекта, выбрав Tools >> [Project Name] Project Properties. Это вызовет диалоговое окно свойств проекта, в которое мы войдем в объявление константы. В поле «Условные аргументы компиляции» введите `[constName] = [value]` . Вы можете ввести более 1 константы, разделив их на двоеточие, например `[constName1] = [value1] : [constName2] = [value2]` .



Предварительно определенные константы

Некоторые константы компиляции уже заранее определены. Какие из них будут зависеть от битности офисной версии, в которой вы используете VBA. Обратите внимание, что Vba7 был представлен вместе с Office 2010 для поддержки 64-разрядных версий Office.

постоянная	16 бит	32-битный	64-битный
VBA6	Ложь	Если Vba6	Ложь
Vba7	Ложь	Если Vba7	Правда
Win16	Правда	Ложь	Ложь
Win32	Ложь	Правда	Правда
Win64	Ложь	Ложь	Правда
макинтош	Ложь	Если Mac	Если Mac

Обратите внимание, что Win64 / Win32 относится к версии Office, а не к версии Windows. Например, Win32 = TRUE в 32-разрядном Office, даже если ОС - это 64-разрядная версия Windows.

Использование Declare Imports, которые работают во всех версиях Office

```
#If Vba7 Then
    ' It's important to check for Win64 first,
```

```
    ' because Win32 will also return true when Win64 does.

    #If Win64 Then
        Declare PtrSafe Function GetFoo64 Lib "exampleLib32" () As LongLong
    #Else
        Declare PtrSafe Function GetFoo Lib "exampleLib32" () As Long
    #End If
#Else
    ' Must be Vba6, the PtrSafe keyword didn't exist back then,
    ' so we need to declare Win32 imports a bit differently than above.

    #If Win32 Then
        Declare Function GetFoo Lib "exampleLib32"() As Long
    #Else
        Declare Function GetFoo Lib "exampleLib"() As Integer
    #End If
#End If
```

Это может быть немного упрощено в зависимости от того, какие версии офиса вам нужны для поддержки. Например, не многие люди по-прежнему поддерживают 16-разрядные версии Office. Последняя версия 16-разрядного офиса - версия 4.3, выпущенная в 1994 году , поэтому для почти всех современных случаев (включая Office 2007) достаточно следующего объявления.

```
#If Vba7 Then
    ' It's important to check for Win64 first,
    ' because Win32 will also return true when Win64 does.

    #If Win64 Then
        Declare PtrSafe Function GetFoo64 Lib "exampleLib32" () As LongLong
    #Else
        Declare PtrSafe Function GetFoo Lib "exampleLib32" () As Long
    #End If
#Else
    ' Must be Vba6. We don't support 16 bit office, so must be Win32.

    Declare Function GetFoo Lib "exampleLib32"() As Long
#End If
```

Если вам не нужно поддерживать что-либо старше Office 2010, это заявление работает очень хорошо.

```
' We only have 2010 installs, so we already know we have Vba7.

#If Win64 Then
    Declare PtrSafe Function GetFoo64 Lib "exampleLib32" () As LongLong
#Else
    Declare PtrSafe Function GetFoo Lib "exampleLib32" () As Long
#End If
```

Прочитайте Условная компиляция онлайн: https://riptutorial.com/ru/vba/topic/3364/условная-компиляция

глава 45: Часто используемые манипуляции с строками

Вступление

Краткие примеры для функций MID LEFT и RIGHT с использованием INSTR FIND и LEN.

Как вы находите текст между двумя поисковыми терминами (Скажите: после двоеточия и перед запятой)? Как вы получаете оставшуюся часть слова (используя MID или используя RIGHT)? Какая из этих функций использует параметры на основе нуля и коды возврата vs One-based? Что происходит, когда что-то идет не так? Как они обрабатывают пустые строки, необоснованные результаты и отрицательные числа?

Examples

Строковые манипуляции часто используются примеры

Лучшие примеры MID () и других примеров извлечения строк, которые в настоящее время отсутствуют в Интернете. Пожалуйста, помогите мне сделать хороший пример или заполнить этот. Что-то вроде этого:

```
DIM strEmpty as String, strNull as String, theText as String
DIM idx as Integer
DIM letterCount as Integer
DIM result as String

strNull = NOTHING
strEmpty = ""
theText = "1234, 78910"

' _________________
' Extract the word after the comma ", "  and before "910"  result: "78" ***
' _________________

' Get index (place) of comma using INSTR
idx = ...   ' some explanation here
if idx < ... ' check if no comma found in text

' or get index of comma using FIND
idx = ...   ' some explanation here... Note: The difference is...
if idx < ...  ' check if no comma found in text

result = MID(theText, ..., LEN(...

' Retrieve remaining word after the comma
result = MID(theText, idx+1, LEN(theText) - idx+1)

' Get word until the comma using LEFT
result = LEFT(theText, idx - 1)
```

```
' Get remaining text after the comma-and-space using RIGHT
result = ...

' What happens when things go wrong
result = MID(strNothing, 1, 2)    ' this causes ...
result = MID(strEmpty, 1, 2) ' which causes...
result = MID(theText, 30, 2) ' and now...
result = MID(theText, 2, 999) ' no worries...
result = MID(theText, 0, 2)
result = MID(theText, 2, 0)
result = MID(theText -1, 2)
result = MID(theText 2, -1)
idx = INSTR(strNothing, "123")
idx = INSTR(theText, strNothing)
idx = INSTR(theText, strEmpty)
i = LEN(strEmpty)
i = LEN(strNothing) '...
```

Не стесняйтесь редактировать этот пример и сделать его лучше. Пока это остается ясным и имеет в нем общие практики использования.

Прочитайте Часто используемые манипуляции с строками онлайн:
https://riptutorial.com/ru/vba/topic/8890/часто-используемые-манипуляции-с-строками

глава 46: Чтение 2GB + файлов в двоичном формате в VBA и хэш файлах

Вступление

Существует простой способ чтения файлов в двоичном формате в VBA, однако он имеет ограничение на 2 ГБ (2 147 483 647 байт - максимальный тип данных Long). По мере развития технологии этот предел 2 ГБ легко нарушается. например, ISO образ операционной системы установить DVD-диск. Microsoft предоставляет возможность преодолеть это с помощью низкоуровневого API Windows, и вот его резервная копия.

Также продемонстрируйте (прочитайте часть) для вычисления `fciv.exe` файлов без внешней программы, такой как `fciv.exe` от Microsoft.

замечания

МЕТОДЫ КЛАССА MICROSOFT

Имя метода	Описание
Открыт	Возвращает логическое значение, указывающее, открыт ли файл.
OpenFile (*sFileNam* e As String)	Открывает файл, указанный аргументом sFileName.
CloseFile	Закрывает открытый файл.
ReadBytes (*ByteCount* As Long)	Читает байты ByteCount и возвращает их в массив байтов Variant и перемещает указатель.
WriteBytes (*DataBytes ()* как байт)	Записывает содержимое массива байтов в текущую позицию в файле и перемещает указатель.
Промывать	Заставляет Windows очищать кеш записи.
SeekAbsolute (*HighPos* As Long, *LowPos* As Long)	Перемещает указатель файла в назначенную позицию с начала файла. Хотя VBA рассматривает DWORDS как подписанные значения, API обрабатывает их как unsigned. Сделать аргумент высокого порядка ненулевым, чтобы он превысил 4 ГБ.

Имя метода	Описание
	Низкоуровневый DWORD будет отрицательным для значений от 2 до 4 ГБ.
SeekRelative (*смещение* как долго)	Перемещает указатель файла до +/- 2 ГБ из текущего местоположения. Вы можете переписать этот метод, чтобы разрешить смещения более 2 ГБ путем преобразования 64-разрядного сдвига со знаком в два 32-битных значения.

СВОИСТВА КЛАССА MICROSOFT

Имущество	Описание
FileHandle	Дескриптор файла для открытого файла. Это несовместимо с файлами VBA.
Имя файла	Имя открытого файла.
AutoFlush	Устанавливает / указывает, будет ли WriteBytes автоматически вызывать метод Flush.

НОРМАЛЬНЫИ МОДУЛЬ

функция	Заметки
GetFileHash (*sFile* As String, *uBlockSize* As Double, *sHashType* As String)	Просто введите полный путь для хэширования, Blocksize для использования (количество байтов) и тип используемого хеша - одну из частных констант: **HashTypeMD5** , **HashTypeSHA1** , **HashTypeSHA256** , **HashTypeSHA384** , **HashTypeSHA512** . Это было сделано как можно более общим.

Вы должны un / комментировать **uFileSize As Double** соответственно. Я тестировал MD5 и SHA1.

Examples

Это должно быть в модуле класса, примеры, которые позже называются «Random»,

```
' How To Seek Past VBA's 2GB File Limit
' Source: https://support.microsoft.com/en-us/kb/189981 (Archived)
' This must be in a Class Module
```

```
Option Explicit

Public Enum W32F_Errors
    W32F_UNKNOWN_ERROR = 45600
    W32F_FILE_ALREADY_OPEN
    W32F_PROBLEM_OPENING_FILE
    W32F_FILE_ALREADY_CLOSED
    W32F_Problem_seeking
End Enum

Private Const W32F_SOURCE = "Win32File Object"
Private Const GENERIC_WRITE = &H40000000
Private Const GENERIC_READ = &H80000000
Private Const FILE_ATTRIBUTE_NORMAL = &H80
Private Const CREATE_ALWAYS = 2
Private Const OPEN_ALWAYS = 4
Private Const INVALID_HANDLE_VALUE = -1

Private Const FILE_BEGIN = 0, FILE_CURRENT = 1, FILE_END = 2

Private Const FORMAT_MESSAGE_FROM_SYSTEM = &H1000

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

Private Declare Function ReadFile Lib "kernel32" ( _
    ByVal hFile As Long, _
    lpBuffer As Any, _
    ByVal nNumberOfBytesToRead As Long, _
    lpNumberOfBytesRead As Long, _
    ByVal lpOverlapped As Long) As Long

Private Declare Function CloseHandle Lib "kernel32" (ByVal hObject As Long) As Long

Private Declare Function WriteFile Lib "kernel32" ( _
    ByVal hFile As Long, _
    lpBuffer As Any, _
    ByVal nNumberOfBytesToWrite As Long, _
    lpNumberOfBytesWritten As Long, _
    ByVal lpOverlapped As Long) As Long

Private Declare Function CreateFile Lib "kernel32" Alias "CreateFileA" ( _
    ByVal lpFileName As String, _
    ByVal dwDesiredAccess As Long, _
    ByVal dwShareMode As Long, _
    ByVal lpSecurityAttributes As Long, _
    ByVal dwCreationDisposition As Long, _
    ByVal dwFlagsAndAttributes As Long, _
    ByVal hTemplateFile As Long) As Long

Private Declare Function SetFilePointer Lib "kernel32" ( _
    ByVal hFile As Long, _
    ByVal lDistanceToMove As Long, _
    lpDistanceToMoveHigh As Long, _
```

```
    ByVal dwMoveMethod As Long) As Long

Private Declare Function FlushFileBuffers Lib "kernel32" (ByVal hFile As Long) As Long

Private hFile As Long, sFName As String, fAutoFlush As Boolean

Public Property Get FileHandle() As Long
    If hFile = INVALID_HANDLE_VALUE Then
        RaiseError W32F_FILE_ALREADY_CLOSED
    End If
    FileHandle = hFile
End Property

Public Property Get FileName() As String
    If hFile = INVALID_HANDLE_VALUE Then
        RaiseError W32F_FILE_ALREADY_CLOSED
    End If
    FileName = sFName
End Property

Public Property Get IsOpen() As Boolean
    IsOpen = hFile <> INVALID_HANDLE_VALUE
End Property

Public Property Get AutoFlush() As Boolean
    If hFile = INVALID_HANDLE_VALUE Then
        RaiseError W32F_FILE_ALREADY_CLOSED
    End If
    AutoFlush = fAutoFlush
End Property

Public Property Let AutoFlush(ByVal NewVal As Boolean)
    If hFile = INVALID_HANDLE_VALUE Then
        RaiseError W32F_FILE_ALREADY_CLOSED
    End If
    fAutoFlush = NewVal
End Property

Public Sub OpenFile(ByVal sFileName As String)
    If hFile <> INVALID_HANDLE_VALUE Then
        RaiseError W32F_FILE_ALREADY_OPEN, sFName
    End If
    hFile = CreateFile(sFileName, GENERIC_WRITE Or GENERIC_READ, 0, 0, OPEN_ALWAYS,
FILE_ATTRIBUTE_NORMAL, 0)
    If hFile = INVALID_HANDLE_VALUE Then
        RaiseError W32F_PROBLEM_OPENING_FILE, sFileName
    End If
    sFName = sFileName
End Sub

Public Sub CloseFile()
    If hFile = INVALID_HANDLE_VALUE Then
        RaiseError W32F_FILE_ALREADY_CLOSED
    End If
    CloseHandle hFile
    sFName = ""
    fAutoFlush = False
    hFile = INVALID_HANDLE_VALUE
End Sub

Public Function ReadBytes(ByVal ByteCount As Long) As Variant
```

```
    Dim BytesRead As Long, Bytes() As Byte
    If hFile = INVALID_HANDLE_VALUE Then
        RaiseError W32F_FILE_ALREADY_CLOSED
    End If
    ReDim Bytes(0 To ByteCount - 1) As Byte
    ReadFile hFile, Bytes(0), ByteCount, BytesRead, 0
    ReadBytes = Bytes
End Function

Public Sub WriteBytes(DataBytes() As Byte)
    Dim fSuccess As Long, BytesToWrite As Long, BytesWritten As Long
    If hFile = INVALID_HANDLE_VALUE Then
        RaiseError W32F_FILE_ALREADY_CLOSED
    End If
    BytesToWrite = UBound(DataBytes) - LBound(DataBytes) + 1
    fSuccess = WriteFile(hFile, DataBytes(LBound(DataBytes)), BytesToWrite, BytesWritten, 0)
    If fAutoFlush Then Flush
End Sub

Public Sub Flush()
    If hFile = INVALID_HANDLE_VALUE Then
        RaiseError W32F_FILE_ALREADY_CLOSED
    End If
    FlushFileBuffers hFile
End Sub

Public Sub SeekAbsolute(ByVal HighPos As Long, ByVal LowPos As Long)
    If hFile = INVALID_HANDLE_VALUE Then
        RaiseError W32F_FILE_ALREADY_CLOSED
    End If
    LowPos = SetFilePointer(hFile, LowPos, HighPos, FILE_BEGIN)
End Sub

Public Sub SeekRelative(ByVal Offset As Long)
    Dim TempLow As Long, TempErr As Long
    If hFile = INVALID_HANDLE_VALUE Then
        RaiseError W32F_FILE_ALREADY_CLOSED
    End If
    TempLow = SetFilePointer(hFile, Offset, ByVal 0&, FILE_CURRENT)
    If TempLow = -1 Then
        TempErr = Err.LastDllError
        If TempErr Then
            RaiseError W32F_Problem_seeking, "Error " & TempErr & "." & vbCrLf & CStr(TempErr)
        End If
    End If
End Sub

Private Sub Class_Initialize()
    hFile = INVALID_HANDLE_VALUE
End Sub

Private Sub Class_Terminate()
    If hFile <> INVALID_HANDLE_VALUE Then CloseHandle hFile
End Sub

Private Sub RaiseError(ByVal ErrorCode As W32F_Errors, Optional sExtra)
    Dim Win32Err As Long, Win32Text As String
    Win32Err = Err.LastDllError
    If Win32Err Then
        Win32Text = vbCrLf & "Error " & Win32Err & vbCrLf & _
        DecodeAPIErrors(Win32Err)
```

```
    End If
    Select Case ErrorCode
        Case W32F_FILE_ALREADY_OPEN
            Err.Raise W32F_FILE_ALREADY_OPEN, W32F_SOURCE, "The file '" & sExtra & "' is
already open." & Win32Text
        Case W32F_PROBLEM_OPENING_FILE
            Err.Raise W32F_PROBLEM_OPENING_FILE, W32F_SOURCE, "Error opening '" & sExtra &
"'." & Win32Text
        Case W32F_FILE_ALREADY_CLOSED
            Err.Raise W32F_FILE_ALREADY_CLOSED, W32F_SOURCE, "There is no open file."
        Case W32F_Problem_seeking
            Err.Raise W32F_Problem_seeking, W32F_SOURCE, "Seek Error." & vbCrLf & sExtra
        Case Else
            Err.Raise W32F_UNKNOWN_ERROR, W32F_SOURCE, "Unknown error." & Win32Text
    End Select
End Sub

Private Function DecodeAPIErrors(ByVal ErrorCode As Long) As String
    Dim sMessage As String, MessageLength As Long
    sMessage = Space$(256)
    MessageLength = FormatMessage(FORMAT_MESSAGE_FROM_SYSTEM, 0&, ErrorCode, 0&, sMessage,
256&, 0&)
    If MessageLength > 0 Then
        DecodeAPIErrors = Left(sMessage, MessageLength)
    Else
        DecodeAPIErrors = "Unknown Error."
    End If
End Function
```

Код для вычисления хеширования файлов в стандартном модуле

```
Private Const HashTypeMD5 As String = "MD5" ' https://msdn.microsoft.com/en-
us/library/system.security.cryptography.md5cryptoserviceprovider(v=vs.110).aspx
Private Const HashTypeSHA1 As String = "SHA1" ' https://msdn.microsoft.com/en-
us/library/system.security.cryptography.sha1cryptoserviceprovider(v=vs.110).aspx
Private Const HashTypeSHA256 As String = "SHA256" ' https://msdn.microsoft.com/en-
us/library/system.security.cryptography.sha256cryptoserviceprovider(v=vs.110).aspx
Private Const HashTypeSHA384 As String = "SHA384" ' https://msdn.microsoft.com/en-
us/library/system.security.cryptography.sha384cryptoserviceprovider(v=vs.110).aspx
Private Const HashTypeSHA512 As String = "SHA512" ' https://msdn.microsoft.com/en-
us/library/system.security.cryptography.sha512cryptoserviceprovider(v=vs.110).aspx

Private uFileSize As Double ' Comment out if not testing performance by FileHashes()

Sub FileHashes()
    Dim tStart As Date, tFinish As Date, sHash As String, aTestFiles As Variant, oTestFile As
Variant, aBlockSizes As Variant, oBlockSize As Variant
    Dim BLOCKSIZE As Double

    ' This performs performance testing on different file sizes and block sizes
    aBlockSizes = Array("2^12-1", "2^13-1", "2^14-1", "2^15-1", "2^16-1", "2^17-1", "2^18-1",
"2^19-1", "2^20-1", "2^21-1", "2^22-1", "2^23-1", "2^24-1", "2^25-1", "2^26-1")
    aTestFiles = Array("C:\ISO\clonezilla-live-2.2.2-37-amd64.iso",
"C:\ISO\HPIP201.2014_0902.29.iso",
"C:\ISO\SW_DVD5_Windows_Vista_Business_W32_32BIT_English.ISO",
"C:\ISO\Win10_1607_English_x64.iso",
"C:\ISO\SW_DVD9_Windows_Svr_Std_and_DataCtr_2012_R2_64Bit_English.ISO")
    Debug.Print "Test files: " & Join(aTestFiles, " | ")
    Debug.Print "BlockSizes: " & Join(aBlockSizes, " | ")
```

```
    For Each oTestFile In aTestFiles
        Debug.Print oTestFile
        For Each oBlockSize In aBlockSizes
            BLOCKSIZE = Evaluate(oBlockSize)
            tStart = Now
            sHash = GetFileHash(CStr(oTestFile), BLOCKSIZE, HashTypeMD5)
            tFinish = Now
            Debug.Print sHash, uFileSize, Format(tFinish - tStart, "hh:mm:ss"), oBlockSize & "
(" & BLOCKSIZE & ")"
        Next
    Next
End Sub

Private Function GetFileHash(ByVal sFile As String, ByVal uBlockSize As Double, ByVal
sHashType As String) As String
    Dim oFSO As Object ' "Scripting.FileSystemObject"
    Dim oCSP As Object ' One of the "CryptoServiceProvider"
    Dim oRnd As Random ' "Random" Class by Microsoft, must be in the same file
    Dim uBytesRead As Double, uBytesToRead As Double, bDone As Boolean
    Dim aBlock() As Byte, aBytes As Variant ' Arrays to store bytes
    Dim aHash() As Byte, sHash As String, i As Long
    'Dim uFileSize As Double ' Un-Comment if GetFileHash() is to be used individually

    Set oRnd = New Random ' Class by Microsoft: Random
    Set oFSO = CreateObject("Scripting.FileSystemObject")
    Set oCSP = CreateObject("System.Security.Cryptography." & sHashType &
"CryptoServiceProvider")

    If oFSO Is Nothing Or oRnd Is Nothing Or oCSP Is Nothing Then
        MsgBox "One or more required objects cannot be created"
        GoTo CleanUp
    End If

    uFileSize = oFSO.GetFile(sFile).Size ' FILELEN() has 2GB max!
    uBytesRead = 0
    bDone = False
    sHash = String(oCSP.HashSize / 4, "0") ' Each hexadecimal has 4 bits

    Application.ScreenUpdating = False
    ' Process the file in chunks of uBlockSize or less
    If uFileSize = 0 Then
        ReDim aBlock(0)
        oCSP.TransformFinalBlock aBlock, 0, 0
        bDone = True
    Else
        With oRnd
            .OpenFile sFile
            Do
                If uBytesRead + uBlockSize < uFileSize Then
                    uBytesToRead = uBlockSize
                Else
                    uBytesToRead = uFileSize - uBytesRead
                    bDone = True
                End If
                ' Read in some bytes
                aBytes = .ReadBytes(uBytesToRead)
                aBlock = aBytes
                If bDone Then
                    oCSP.TransformFinalBlock aBlock, 0, uBytesToRead
                    uBytesRead = uBytesRead + uBytesToRead
                Else
```

```
                    uBytesRead = uBytesRead + oCSP.TransformBlock(aBlock, 0, uBytesToRead,
aBlock, 0)
                End If
                DoEvents
            Loop Until bDone
            .CloseFile
        End With
    End If
    If bDone Then
        ' convert Hash byte array to an hexadecimal string
        aHash = oCSP.hash
        For i = 0 To UBound(aHash)
            Mid$(sHash, i * 2 + (aHash(i) > 15) + 2) = Hex(aHash(i))
        Next
    End If
    Application.ScreenUpdating = True
    ' Clean up
    oCSP.Clear
CleanUp:
    Set oFSO = Nothing
    Set oRnd = Nothing
    Set oCSP = Nothing
    GetFileHash = sHash
End Function
```

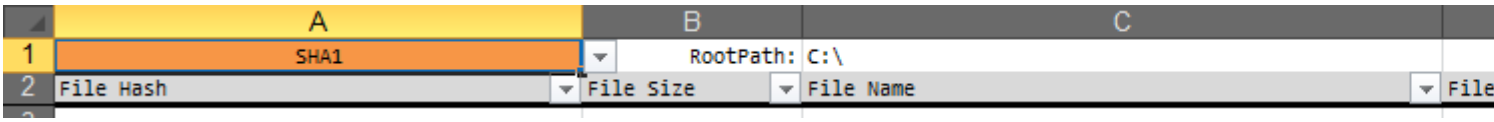
Результат довольно интересный, мои тестовые файлы показывают, что `BLOCKSIZE = 131071` **(2 ^ 17-1)** дает общую лучшую производительность с 32-битным Office 2010 на Windows 7 x64, а лучше всего *2 ^ 16-1 (65535)* . Примечание `2^27-1` дает *выход из памяти* .

Размер файла (байты)	Имя файла
146800640	Clonezilla-жить-2.2.2-37-amd64.iso
798210048	HPIP201.2014_0902.29.iso
2073016320	SW_DVD5_Windows_Vista_Business_W32_32BIT_English.ISO
4380387328	Win10_1607_English_x64.iso
5400115200	SW_DVD9_Windows_Svr_Std_and_DataCtr_2012_R2_64Bit_English.ISO

Вычисление всех файлов Hash из корневой папки

Другой вариант кода выше дает вам большую производительность, если вы хотите получить хэш-коды всех файлов из корневой папки, включая все подпапки.

Пример рабочего листа:



Код

```
Option Explicit

Private Const HashTypeMD5 As String = "MD5" ' https://msdn.microsoft.com/en-
us/library/system.security.cryptography.md5cryptoserviceprovider(v=vs.110).aspx
Private Const HashTypeSHA1 As String = "SHA1" ' https://msdn.microsoft.com/en-
us/library/system.security.cryptography.sha1cryptoserviceprovider(v=vs.110).aspx
Private Const HashTypeSHA256 As String = "SHA256" ' https://msdn.microsoft.com/en-
us/library/system.security.cryptography.sha256cryptoserviceprovider(v=vs.110).aspx
Private Const HashTypeSHA384 As String = "SHA384" ' https://msdn.microsoft.com/en-
us/library/system.security.cryptography.sha384cryptoserviceprovider(v=vs.110).aspx
Private Const HashTypeSHA512 As String = "SHA512" ' https://msdn.microsoft.com/en-
us/library/system.security.cryptography.sha512cryptoserviceprovider(v=vs.110).aspx

Private Const BLOCKSIZE As Double = 131071 ' 2^17-1

Private oFSO As Object
Private oCSP As Object
Private oRnd As Random ' Requires the Class from Microsoft https://support.microsoft.com/en-
us/kb/189981
Private sHashType As String
Private sRootFDR As String
Private oRng As Range
Private uFileCount As Double

Sub AllFileHashes() ' Active-X button calls this
    Dim oWS As Worksheet
    ' | A: FileHash | B: FileSize | C: FileName | D: FilaName and Path | E: File Last
Modification Time | F: Time required to calculate has code (seconds)
    With ThisWorkbook
        ' Clear All old entries on all worksheets
        For Each oWS In .Worksheets
            Set oRng = Intersect(oWS.UsedRange, oWS.UsedRange.Offset(2))
            If Not oRng Is Nothing Then oRng.ClearContents
        Next
        With .Worksheets(1)
            sHashType = Trim(.Range("A1").Value) ' Range(A1)
            sRootFDR = Trim(.Range("C1").Value) ' Range(C1) Column B for file size
            If Len(sHashType) = 0 Or Len(sRootFDR) = 0 Then Exit Sub
            Set oRng = .Range("A3") ' First entry on First Page
        End With
    End With

    uFileCount = 0
    If oRnd Is Nothing Then Set oRnd = New Random ' Class by Microsoft: Random
    If oFSO Is Nothing Then Set oFSO = CreateObject("Scripting.FileSystemObject") ' Just to
get correct FileSize
    If oCSP Is Nothing Then Set oCSP = CreateObject("System.Security.Cryptography." &
sHashType & "CryptoServiceProvider")

    ProcessFolder oFSO.GetFolder(sRootFDR)

    Application.StatusBar = False
    Application.ScreenUpdating = True
```

```
    oCSP.Clear
    Set oCSP = Nothing
    Set oRng = Nothing
    Set oFSO = Nothing
    Set oRnd = Nothing
    Debug.Print "Total file count: " & uFileCount
End Sub

Private Sub ProcessFolder(ByRef oFDR As Object)
    Dim oFile As Object, oSubFDR As Object, sHash As String, dStart As Date, dFinish As Date
    Application.ScreenUpdating = False
    For Each oFile In oFDR.Files
        uFileCount = uFileCount + 1
        Application.StatusBar = uFileCount & ": " & Right(oFile.Path, 255 - Len(uFileCount) -
2)
        oCSP.Initialize ' Reinitialize the CryptoServiceProvider
        dStart = Now
        sHash = GetFileHash(oFile, BLOCKSIZE, sHashType)
        dFinish = Now
        With oRng
            .Value = sHash
            .Offset(0, 1).Value = oFile.Size ' File Size in bytes
            .Offset(0, 2).Value = oFile.Name ' File name with extension
            .Offset(0, 3).Value = oFile.Path ' Full File name and Path
            .Offset(0, 4).Value = FileDateTime(oFile.Path) ' Last modification timestamp of
file
            .Offset(0, 5).Value = dFinish - dStart ' Time required to calculate hash code
        End With
        If oRng.Row = Rows.Count Then
            ' Max rows reached, start on Next sheet
            If oRng.Worksheet.Index + 1 > ThisWorkbook.Worksheets.Count Then
                MsgBox "All rows in all worksheets have been used, please create more sheets"
                End
            End If
            Set oRng = ThisWorkbook.Sheets(oRng.Worksheet.Index + 1).Range("A3")
            oRng.Worksheet.Activate
        Else
            ' Move to next row otherwise
            Set oRng = oRng.Offset(1)
        End If
    Next
    'Application.StatusBar = False
    Application.ScreenUpdating = True
    oRng.Activate
    For Each oSubFDR In oFDR.SubFolders
        ProcessFolder oSubFDR
    Next
End Sub

Private Function GetFileHash(ByVal sFile As String, ByVal uBlockSize As Double, ByVal
sHashType As String) As String
    Dim uBytesRead As Double, uBytesToRead As Double, bDone As Boolean
    Dim aBlock() As Byte, aBytes As Variant ' Arrays to store bytes
    Dim aHash() As Byte, sHash As String, i As Long, oTmp As Variant
    Dim uFileSize As Double ' Un-Comment if GetFileHash() is to be used individually

    If oRnd Is Nothing Then Set oRnd = New Random ' Class by Microsoft: Random
    If oFSO Is Nothing Then Set oFSO = CreateObject("Scripting.FileSystemObject") ' Just to
get correct FileSize
    If oCSP Is Nothing Then Set oCSP = CreateObject("System.Security.Cryptography." &
sHashType & "CryptoServiceProvider")
```

```
    If oFSO Is Nothing Or oRnd Is Nothing Or oCSP Is Nothing Then
        MsgBox "One or more required objects cannot be created"
        Exit Function
    End If

    uFileSize = oFSO.GetFile(sFile).Size ' FILELEN() has 2GB max
    uBytesRead = 0
    bDone = False
    sHash = String(oCSP.HashSize / 4, "0") ' Each hexadecimal is 4 bits

    ' Process the file in chunks of uBlockSize or less
    If uFileSize = 0 Then
        ReDim aBlock(0)
        oCSP.TransformFinalBlock aBlock, 0, 0
        bDone = True
    Else
        With oRnd
            On Error GoTo CannotOpenFile
            .OpenFile sFile
            Do
                If uBytesRead + uBlockSize < uFileSize Then
                    uBytesToRead = uBlockSize
                Else
                    uBytesToRead = uFileSize - uBytesRead
                    bDone = True
                End If
                ' Read in some bytes
                aBytes = .ReadBytes(uBytesToRead)
                aBlock = aBytes
                If bDone Then
                    oCSP.TransformFinalBlock aBlock, 0, uBytesToRead
                    uBytesRead = uBytesRead + uBytesToRead
                Else
                    uBytesRead = uBytesRead + oCSP.TransformBlock(aBlock, 0, uBytesToRead,
aBlock, 0)
                End If
                DoEvents
            Loop Until bDone
            .CloseFile
CannotOpenFile:
            If Err.Number <> 0 Then ' Change the hash code to the Error description
                oTmp = Split(Err.Description, vbCrLf)
                sHash = oTmp(1) & ":" & oTmp(2)
            End If
        End With
    End If
    If bDone Then
        ' convert Hash byte array to an hexadecimal string
        aHash = oCSP.hash
        For i = 0 To UBound(aHash)
            Mid$(sHash, i * 2 + (aHash(i) > 15) + 2) = Hex(aHash(i))
        Next
    End If
    GetFileHash = sHash
End Function
```

Прочитайте Чтение 2GB + файлов в двоичном формате в VBA и хэш файлах онлайн: https://riptutorial.com/ru/vba/topic/8786/чтение-2gb-plus-файлов-в-двоичном-формате-в-vba-и-хэш-файлах

кредиты

S. No	Главы	Contributors
1	Начало работы с VBA	0m3r, Andre Terra, Benno Grimm, Bookeater, Comintern, Community, Derpcode, Kaz, lfrandom, litelite, Maarten van Stam, Macro Man, Máté Juhász, Nick Dewitt, PankajKushwaha, RubberDuck, Stefan Pinnow
2	API-запросы	paul bica
3	CreateObject vs. GetObject	Branislav Kollár, Dave, Tim
4	Scripting.FileSystemObject	Comintern, Dave, Macro Man, Mikegrann, RubberDuck, Siva, Steve Rindsberg, ThunderFrame
5	Автоматизация или использование других приложений Библиотеки	Branislav Kollár
6	Атрибуты	hymced, Mat's Mug, RamenChef, RubberDuck
7	Измерение длины строк	Steve Rindsberg, ThunderFrame
8	Интерфейсы	Neil Mussett
9	Ключевое слово Option VBA	Jeeped, Maarten van Stam, Macro Man, Mat's Mug, RamenChef, RubberDuck, Stefan Pinnow, Thomas G, ThunderFrame
10	Коллекции	Comintern
11	Комментарии	Comintern, Hosch250, Johnny C, litelite, Macro Man, Nijin22, Shawn V. Wilson, ThunderFrame
12	Конкатенация строк	ThunderFrame
13	Копирование, возврат и передача массивов	Mark.R
14	Макрозащита и подписание VBA-проектов / -модулей	0m3r

15	Манипуляция времени по времени	Comintern, FreeMan, Thomas G
16	Массивы	Comintern, Dave, Hubisan, jamheadart, Josan Iracheta, Maarten van Stam, Mark.R, Mat's Mug, Miguel_Ryu, Tazaf
17	Назначение строк с повторяющимися символами	ThunderFrame
18	Нелатинские символы	Neil Mussett
19	Обработка ошибок	Comintern, Logan Reed, Mat's Mug
20	Объект Scripting.Dictionary	Comintern, Jeeped, Kyle, RamenChef, Tim, Wolf, Zev Spitz
21	Объектно-ориентированный VBA	IvenBach, Mat's Mug
22	Объявление и назначение строк	Comintern, ThunderFrame
23	Объявление переменных	Comintern, dadde, Dave, Franck Dernoncourt, Jeeped, Kaz, lfrandom, litelite, Macro Man, Mark.R, Mat's Mug, Neil Mussett, RubberDuck, Shawn V. Wilson, SWa, Thierry Dalon, ThunderFrame, Tom, Victor Moraes, Zaider
24	операторы	Comintern, Macro Man
25	Ошибки времени выполнения VBA	Branislav Kollár, Macro Man, Mat's Mug
26	Передача аргументов ByRef или ByVal	Branislav Kollár, Comintern, Mat's Mug, R3uK, RamenChef, ZygD
27	Подстроки	Mat's Mug, ThunderFrame
28	Поиск в строках для наличия подстрок	ThunderFrame
29	Пользовательские формы	Mat's Mug
30	Преобразование других типов в строки	ThunderFrame
31	Процедурные вызовы	Macro Man, Mat's Mug, Neil Mussett, Sam Johnson

32	Работа с ADO	Comintern, SandPiper, Tazaf
33	Работа с файлами и каталогами без использования FileSystemObject	Comintern, Macro Man, SandPiper
34	Рекурсия	Mat's Mug, ThunderFrame
35	События	Mat's Mug
36	Соглашения об именах	FreeMan, Kaz, Mat's Mug, Victor Moraes
37	Создание пользовательского класса	Branislav Kollár, Macro Man, Mat's Mug, Neil Mussett, ThunderFrame
38	Создание процедуры	Comintern, LiamH, Mat's Mug, Sivaprasath Vadivel, Tomas Zubiri
39	Сортировка	Neil Mussett
40	Строковые литералы - экранирование, непечатаемые символы и линейные продолжения	Comintern, ThunderFrame
41	Структуры данных	Blackhawk
42	Структуры контроля потока	Benno Grimm, Comintern, Kelly Tessena Keck, Leviathan, litelite, Macro Man, Martin, Mat's Mug, Roland, Siva, ThunderFrame
43	Типы данных и ограничения	Comintern, FreeMan, Neil Mussett, StackzOfZtuff, Stephen Leppik, ThunderFrame
44	Условная компиляция	Macro Man, Mat's Mug, RubberDuck, Steve Rindsberg
45	Часто используемые манипуляции с строками	pashute
46	Чтение 2GB + файлов в двоичном формате в VBA и хэш файлах	PatricK