Coding guidelines

For Visual Basic 5/6 and VBA for Office

Naming convention

©2002-2014 devinfo.net, - Développement Informatique Services
Francesco Foti
http://www.devinfo.net

Document history

When	Who	Version	What
01/21/2014	FOF	1.0	First publication of our guidelines for VB/A.

Foreword

I know, there are some inconsistencies in my naming convention and I do not always seem to follow it hundred percent. The fact is that you're looking at VB code that has quite evolved since 2003 (not to speak of the programmer himself).
With time, some of my coding habits have also changed, so I'll present here only the rules I stick with today, and when it may be of any concern, I'll also briefly expose and talk about some of the inconsistencies you'll encounter in the sources.

Naming conventions

Constants

My preference

- All constant names are in upper case and if they consist of multiple "words" they *may* use an underscore to separate them:
  Examples:
```
'Like API constants:
Private Const SW_MAXIMIZE As Long = 3&
Private Const SW_NORMAL   As Long = 1&
Private Const SW_HIDE     As Long = 0&
Private Const SW_SHOW     As Long = 5&
'And we may do:
Private Const COLFLAG_DATETIME      As Long = 1&
'Which I prefer over:
Private Const COL_FLAG_DATETIME     As Long = 1&
```
- All constants should declared with their type (As …)
  Examples:
```
'DON'T DO THAT:
Private Const SW_MAXIMIZE = 3
'BUT INSTEAD, DO:
Private Const SW_MAXIMIZE As Long = 3&
```

Other conventions

Unfortunately, in my coding career I run into another convention for naming constants, which I sadly adopted, although for a short while. I deeply regret it today and unfortunately, it still stains my code, that's why I have to explain it here.
The convention is fairly simple: constant names always begin with "k" (lower case) and their name is then in CamelCaps.
Examples:
```
'I find that a disgusting way to name constants
Public Const klSizeOfLong      As Long = 4&
Public Const klSizeOfInt       As Long = 2&
Public Const klSizeOfBool      As Long = 2&
Public Const klSizeOfByte      As Long = 1&
'And I strongly prefer that:
Public Const SIZEOF_LONG      As Long = 4&
Public Const SIZEOF_INT       As Long = 2&
Public Const SIZEOF_BOOL      As Long = 2&
Public Const SIZEOF_BYTE      As Long = 1&
```
Over time, all the current code using this old convention will be updated.

Variables

Scope

All variables, variables of an enumerated type, and variable of a custom defined type (structs=types in VB), are prefixed with 1 (one) letter that indicates the variable scope, except for the variables declared in a procedure or function.
There is no underscore following the 1 letter prefix, as opposed as what you may accustomed to see in other language conventions like C++, something I personally dislike.
So, for global variables the prefix letter is “g”, and for module or class module level variables, the prefix letter is “m”.

Data type naming convention for VB

Data type	Prefix
Integer	i
Long	l (minus L) or sometimes i is loosely used when confusion wouldn’t matter so much.
Date	dt
Variant	v
String	s
Boolean	f (and not b as b is for byte)
Byte	b
Single	sng
Double	dbl
Currency	cur or c
Object (or any class instance)	o

So, for any other data type rarely encountered, a bit of common sense is enough; as an example, an error data type would be prefixed by “err”. Another common example is “cn” for a database connection object (“cnDatabase” as an example).

Arrays and Types

When we have an array of any type, an “a” prefix comes before the data type prefix, but after the scope prefix.
When a variant variable is used to hold an array of any type, it *may* also be prefixed with “a”.

Examples

```
‘An array of byte declaration in three different scopes, in a code module:
Public gabGlobal() as Byte
Private mabModule(1 To 1024) as Byte

Sub Test()
  Dim abData() as Byte
  ‘…
End Sub

Sub DeclareDemo()
  Dim iItem    as Integer
  Dim lAPIRet  as Long
  Dim vData    as Variant
  Dim curTotal as Currency
  ‘etc...
End Sub
```

Functions and procedures

- Sub and Function names in CamelCaps.
- Always declare the return type for the function, even if it's a Variant.
- Never return an array, pass it by reference. Now, I know, many developers do that, they return arrays (although this came as a new feature in VB6 / VBA, but is not possible in VB5), but I find that horribly confusing, especially when returned array entries are meant as a way to return multiple distinct values; imho it is best to use type definitions, pass the arrays by reference or revisit the API then.

Function parameters

So this is a cardinal point, a rule that I now also apply to almost every other language I code in.

Every function / sub parameter variable is prefixed with a "p". That is often an invaluable indicator in the body of the function's code that allows to quickly identify what variable has been passed to the function, among the ones that have been declared in its body. Without this prefix, it may be sometimes very difficult to distinguish between the two kinds of variables in the function's code.

It is also quite common to see the prefix "Ret" inserted after the variable type prefix, when a parameter variable is going to be returned by the function.

Declaration example:

```
Public function FindCustomer( _
  Byval psSearchFirstName as string,
  Byval psSearchLastName as string,
  Byref plRetCustomerID as Long) _
  As Boolean
```

Other rules

Code indentation

Indentation is always done with 2 (and not 4) spaces, no tabs (or be sure to set the tab space to 2 spaces also).

Error trapping and handling

In almost all code and class modules, I have private variables to hold any encountered error and context in a function (respectively member function) to access them in read only mode.

Example in the MADOAPI.bas module:

```
'Error context
Private mlErr       As Long
Private msErr       As String
Private msErrCtx    As String

Private Sub ClearErr()
  mlErr = 0&
  msErr = ""
End Sub

Private Sub SetErr(ByVal psErrCtx As String, ByVal plErr As Long, ByVal psErr As
String)
```

```
  msErrCtx = psErrCtx
  mlErr = plErr
  msErr = psErr
End Sub

Public Function ADOLastErr() As Long
  ADOLastErr = mlErr
End Function

Public Function ADOLastErrDesc() As String
  ADOLastErrDesc = msErr
End Function

Public Function ADOLastErrCtx() As String
  ADOLastErrCtx = msErrCtx
End Function
```

Error trapping in a function or Sub is implemented only when it makes sense to do it. It is otherwise left to the responsibility of the calling code to cope with any casual error.

When implemented, error trapping almost always take a similar form as this example:

```
Function AnyThingThatCouldFail() As Boolean
  Dim fOK    As Boolean

  On error Goto AnyThingThatCouldFail_Err

  fOK = True

AnyThingThatCouldFail_Exit:
  'Now would be a good time to set dynamically allocated object to nothing

  AnyThingThatCouldFail = fOK
  Exit Function

AnyThingThatCouldFail_Err:
  SetErr "AnyThingThatCouldFail", Err.number, Err.Description
  Resume AnyThingThatCouldFail_Exit
End Function
```