VBA Code Guidelines

- General Advice
- Parameters
- General errors
- Variables
  - General
  - Declaring
  - Comments
  - Variants
  - Dates
- General Naming Conventions
  - General
  - Prefix
  - Tag
  - Base Name
  - Qualifiers
  - Arrays
  - Constants
- API Declaration
  - Use unique alias names
- Form, Class & Module Naming
  - Internal Naming
  - File naming
  - Object instance naming
  - Notes
- Naming Procedures/Functions/Parameters
  - Function Names
  - Function return values
  - Parameters
- Naming Controls
  - Introduction
  - Control tags
  - Naming menu items
- Naming Data Access Objects
  - ADO
  - ADO objects
  - MS Access objects
- Layout
  - Indentation – tab w idth
  - Indentation - general
- Commenting Code
  - Comments
  - Commenting code w hen doing maintenance w ork
  - Etiquette w hen commenting code
  - Pre-compilation commands
- Error Handling
  - Generic error handler
  - Error handling labels

General Advice

Alw ays use Option Explicit as the first line in every code module. To sw itch this on automatically check Require Variable Declaration in Tools>Options>Editor.

Parameters

Avoid confusion over ByVal and ByRef. Be aw are of the default for parameters being ByRef. Be explicit w hen passing parameters.

Example:

```
Public Sub Load(ByVal strName As String, ByVal strPhone As String)
```

General errors

Error handling must be used w herever practicable i.e. w ithin each procedure.
Use On Error Goto ErrHandler
Handle errors w here they occur. This may involve handling the error and raising it to the client code.

Variables

General

Where global variables are used, they must all be defined in one module.

Declaring

Variables must be dimensioned on separate lines, and should specify a datatype (except w here this is not possible – as w hen using certain scripting languages).

Comments

All variables must be declared at the top of each procedure or module and must ideally be grouped so that all variable types are placed together.

Variants

Variants may be used w here appropriate (e.g. to hold arrays returned by a function, or w here Nulls may be encountered), but alternative data types should be used w here possible.

Dates

Where dates are displayed to users you should avoid ambiguous formats w here either years or days vs. months might be confused (such as DD/MM/YY), how ever the ultimate decision maker on this issue is the customer.

Where dates are being handled “behind the scenes” care should be taken to avoid UK/US format confusion. Particular care should be taken w hen including UK-format dates in literal SQL strings (w here the target Microsoft application may expect dates to be in US format). Where there is the slightest possibility of doubt pass the year, month and day parts separately into DateSerial, of format them in the universally acceptable ISO format YYYY-MM-DD.

General Naming Conventions

General

Object names are made up of four parts:
prefix
tag
base name
qualifier
The four parts are assembled as follows:
[prefixes]tag[BaseName][Qualifier]
Note: The brackets denote that these components are optional and are not part of the name.

Prefix

Prefixes and tags are always lowercase so your eye goes past them to the first uppercase letter where the base name begins. This makes the names more readable. The base and qualifier components begin with an uppercase letter.

Prefix	Use	Notes
None	Local to procedure	No scope prefix as in: dblMaximum
m_	Module level scope	m_strPolicyHolder
g_	Global scope	g_intCarsLast

Tag

The tag is the only required component, but in almost all cases the name will have the base name component since you need to be able to distinguish two objects of the same type.

Variable type	Tag	Notes
Boolean	bln	blnFound
Byte	byt	bytRasterData
Currency	cur	curRevenue
Date (Time)	dat	datStart
Double	dbl	dblTolerance
Enum	enm	enmColours
Integer	int	intQuantity
Long	lng	lngDistance
Single	sng	sngAverage
String	str	strFName
User-defined type	udt	udtEmployee
Variant	var	varCheckSum

Base name

The base name succinctly describes the object, not its class. That is, a base name for a variable for an invoice form must be InvoiceEntry not InvoiceForm as the tag will describe the object. The base name is composed in the form Noun[Verb]. For example, in the variable name blnColourMatch "ColourMatch" is the base name. Naming variables in this way helps to keep them grouped together in documentation and cross-referencing because they will be sorted together alphabetically.

Qualifiers

Object qualifiers may follow a name and further clarify names that are similar. Continuing with our previous example, if you kept two indexes to an array, one for the first item and one for the last this entails two qualified variables such as intColourMatchFirst and intColourMatchLast. Other examples of qualifiers:

Usage	Qualifier	Example
Current element of set	Cur	intCarsCur
First element of set	First	intCarsFirst
Last element of set	Last	intCarsLast
Next element of set	Next	strCustomerNext
Previous element of set	Prev	strCustomerPrev
Lower limit of range	Min	strNameMin
Upper limit of range	Max	strNameMax
Source	Src	lngBufferSrc
Destination	Dest	lngBufferDest

Arrays

Array names must be prefixed with "a". The upper and lower bounds of the array must be declared explicitly (unless they're not known at design-time).
Example:

```
Dim astrMonths(1 To 12) as String
```

Constants

Each word must be capitalised and the words separated with an underscore. The base name must be a description of what the constant represents.

Example:

```
User defined constant: g_intERR_INVALID_NAME
Visual Basic: vbArrowHourglass
```

API Declaration

API declarations must be laid out so that they are easily readable on the screen.

```
Public Declare Function WritePrivateProfileString Lib "kernel32" Alias "WritePrivateProfileStringA" _
    (ByVal lpApplicationName As String, _
    ByVal lpKeyName As Any, _
    ByVal lpString As Any, _
    ByVal lpFileName As String) As Long
```

Use unique alias names

In VB you can call external procedures in DLLs when you know the entry point (the name of the function in the DLL). However, the caveat is that you can only declare the external procedure once. If you load a library that calls the same Windows API that your module calls, you will get the infamous error, “Tried to load module with duplicate procedure definition.”

```
Declare smg_GetActiveWindow Lib "Kernel" Alias _
    "GetActiveWindo" () As Integer
```

Form, Class & Module Naming

Internal Naming

(i.e. the name assigned to the module within the VB Properties)

Module Type	Prefix	Example
Form	frm	frmLogon
Standard module	mod	modUtilities
Class module	C	CPerson
Collection class	C	CPersons1
Interface class	I	IPerson

File naming

(i.e. the name assigned to the module when saving the physical file)

Module Type	Prefix	Example
Form	frm	frmLogon.frm
Standard module	mod	modUtilities.bas
Class module	C	CPerson.cls
Collection class	C	CPersons.cls1

Interface class	I	IPerson.cls

Object instance naming

(i.e. the name assigned when declaring a variable based on the form or class)

Instance of	Prefix	Example
Form	frm	frmLogon
Class	obj	objPerson
Collection	obj	objPersons

Notes

1 Classes which hold collections should have the same “C” prefix as any other classes, but should have a plural name (based on the type of objects held in the collection. E.g. a class to hold a single person would be named CPerson, whereas a collection of Person objects would be named CPersons.

Naming Procedures/Functions/Parameters

Function Names

Tags should not be prefixed to Function or Sub names, but **should** be appended to the parameters of these routines. For example:

Correct approach for internal function:

```
Private Function TotalUp(ByVal sngSubTotal As Single) As Integer
```

Function return values

Function return values should usually be held in a temporary variable and then assigned to the function variable at the end of the routine. This has two benefits. The code is not specific to the name of the function so portability is aided when cutting and pasting part of the function code elsewhere; also the value of the function variable may be used in calculations, otherwise a recursive call would be generated.
Example:

```
Private Function Example(ByVal argintA as Integer) as Single
    Dim sngRetVal as Single

    ' Set default value
    sngRetVal = 0

    <code block>

    ' Set the Function value
    Example = sngRetVal
End Function
```

Parameters

Should you find it useful, you may also prefix parameter names with arg to avoid confusion between variables passed as parameters and those local to the subroutine .
Example:

```
Private Function DoSomething(ByVal argstrMessage as String) as String
```

However, should you choose to adopt this standard it must be applied consistently across the entire project

Naming Controls

Introduction

Controls must be named with uniform prefixes strictly adhering to the following list.

Control tags

Object Type	Tag	Notes
Check box	chk	chkReadOnly
Combo box, drop-down list box	cbo	cboEnglish
Command button	cmd	cmdExit
Common dialog	dlg	dlgFileOpen
Control	ctl	ctlCurrent
Form	frm	frmEntry
Frame	fra	fraLanguage
Grid	grd	grdPrices
Image	img	imgIcon
Key status	key	keyCaps
Label	lbl	lblHelpMessage
Line	lin	linVertical
List box	lst	lstPolicyCodes
Menu	mnu	mnuFileOpen
Report	rpt	rptQtr1Earnings
Shape	shp	shpCircle
Text box	txt	txtLastName
True DBGrid	tdbg	tdbgRecords

Timer	tmr	tmrAlarm
ImageList	ils	ilsAllIcons
Toolbar	tlb	tlbActions
TabStrip	tab	tabOptions
ListView	lvw	lvwHeadings

Naming menu items

The number of menu options can be great, so it is recommended that there be a standard for the names given to menus. The tag for any menu item whether an option or title is mnu. Prefixing must ideally continue beyond the initial prefix. The first prefix after mnu is the menu bar title followed by the option then any subsequent option.

Example:

```
Top level menu item – mnuFile
Menu sub item – mnuFileSave
```

Naming Data Access Objects

ADO

If you include references to both ADO and DAO in the same project you must explicitly specify which object model you wish to use when declaring variables.
Example:

```
Dim cnnStore As ADODB.Connection
Dim cnnOther As DAO.Connection
```

ADO objects

Object Type	Tag	Example
Command	cmd	cmdBooks
Connection	cnn	cnnLibrary
Parameter	prm	prmTitle
Error	err	errLoop
Recordset	rst	rstForecast

MS Access objects

The following is a suggested naming convention for use with MS Access objects – you may find it useful for larger Access

projects which have many objects within the same database.

Object Type	Tag	Example
Table	tbl	tblCustomer
Query (select)	qry	qryOverAchiever
Query (append)	qapp	qappNew Product
Query (crosstab)	qxtb	qxtbRegionSales
Query (delete)	qdel	qdelOldAccount
Query (make table)	qmak	qmakShipTo
Query (update)	qupd	qupdDiscount
Form	frm	frmCustomer
Form (dialog)	fdlg	fdlgLogin
Form (message)	fmsg	fmsgWait
Form (subform)	fsub	fsubOrder
Report	rpt	rptInsuranceValue
Report (subreport)	rsub	rsubOrder
Macro (menu)	mmnu	mmnuEntryFormFile
Module	mod	modBilling

Layout

Indentation – tab width

When working in a VB or VBA design environment you **must** have the **Tab Width** set to 4 (see the Editor tab in Tools > Options). This is the default VB setting, and using it ensures compatibility when code is worked on by more than one person.

Indentation - general

Code must be indented consistently adhering to the following rules:

- Declarations must not be indented.
- On Error statements and line labels/numbers must not be indented.
- Start code indented to one tab stop.
- Code within If-Else-EndIf, For-Next, Do While/Until and any other loops must be indented a further tab stop within the body.
- Code between add/edit and update statements must be indented a further tab stop.
- Case statements must be indented to one stop after the Select Case. Code following the Case statements must be indented a further Tab stop.
- Code between With and End With statements must be indented by one tab stop.
- Code within error trap must be indented by to one tab stop.
  Example

```
Dim strTest as String
Dim wrk as Workspace
On Error Goto ErrHandler
    If strTest = "" Then
        strTest = "Nothing"
    Else
        strTest = ""
    EndIf

    Do While Not rst.EOF
        rst.Add
        rst(0) = strTest
        rst.Update
    Loop

    Select Case strTest
        Case ""
            <code block>
        Case Else
            <code block>
        End Select
ExitHere:
    Exit Sub
ErrHandler:
    Resume ExitHere
```

Commenting Code

Comments

Remember the following points:

- Code must be commented appropriately. The goal should be to improve understanding and maintainability of the code.
- Comments should explain the reasoning behind the code. It may be obvious to the original developer what a piece of code does but somebody reading it may have no idea why it has to be there. When you write a piece of code, imagine someone else having to read through it 3 months later. Will it make sense to them?
- Important variable declarations may include an inline comment describing the use of the variable being declared.

Example:

```
Dim strLookUp as String 'Accepts value from user to search for
```

- Comments for individual lines appear above, or of the code to which they refer.
- The functional overview comment of a procedure may be indented one space to aid readability.

Example:

```
Public Sub DeleteCustomer(ByVal argintID As Long)
    'Removes customer from Database
    cnVideo.Execute "DELETE FROM Customer WHERE CustomerID=" & argintID
End Sub
```

Commenting code when doing maintenance work

Avoid over-commenting code when doing maintenance work. Bear in mind the need to maintain overall clarity in the code, and remember that revision history should be taken care of by SourceSafe

Make sure that any existing comments still make sense **after** you've made your changes - paying particular attention to any comments/explanations in the header of the routine.
You are responsible for ensuring that **all** existing comments remain accurate (and that they still make sense) after your changes have been implemented.
Although SourceSafe controls the history, It is handy to future users if new blocks of code are commented with the date, initials of developer and the CR number to aim future developers reading the code.

Etiquette when commenting code

When you include one or more routines written by other developers in your project you should ensure that any author (and assumption/purpose) information in the routine header is kept accurate. You should probably retain the original author's name, but you **must** also include your own name if you have changed it in any way at all.

Pre-compilation commands

These are treated as a code IF statement would be. All code relating to the condition must be indented as if it was a normal IF block. These can be useful for including/excluding debug code etc. For example:

```
#Const DebugMode = True
#IF  DebugMode THEN
    <code block>
#ELSE
    <code block>
#ENDIF
```

Error Handling

Generic error handler

Consistent error handlers must be implemented. The following error handler should be used:

```
On Error GoTo ErrHandler
    <code block>

ExitHere:
On Error Resume Next
       <code block>
Exit Sub
ErrHandler:
    [WriteErrLog Err.Number]
       Select Case Err.Number
          Case <Err No>
                 Resume Next
          Case <Err No>
                 Resume ExitHere
          Case Else
                 ' Unexpected Error
                 Resume ExitHere
       End Select
End Sub
```

Error handling labels

The labels **ErrHandler** and **ExitHere** are used both for consistency across routines, and to facilitate easier copying and pasting

of error handlers betw een routines.