GENESYS SOLUTIONS

Genesys Coding Standard for Visual Basic

Dave Carrington
Research & Development

Contents

This Coding Standard has been adapted from the web page entitled *Visual Basic Coding Standards*, which can be found at http://www.gui.com.au/jkcoding.htm.

1 File Organisation

1.1 Beginning Comments

At the beginning of every Module, there must be a comment of the following format:

```
' Name:               The name of the Module
' Author(s):         All who contributed to this Module
' Date:              Date the Module was last altered
' Version Number:    The version number of this update.
'                    Using the standard major/minor revision system
'                    Starting from 1.0
' Description:       What the Module does.  If it becomes too long
'                    consider breaking it down into smaller
'                    components.
' Changes History:   List of changes, referenced by version number
'                    outlining changes from previous version
'                    i.e.
'                           1.1     fixed bug that caused program to
'                                   crash.
'                           1.2     added the blah functionality.
```

1.2 Procedure & Function Comments

Before each Procedure or Function, the following style of comment should be added:

```
' Name:        The Procedure or Function name.
' Author(s):   The name of all authors who have contributed to
'              this Procedure or Function.  Only include if there is
'              more than one author for this Module.
' Description: Brief description of what the Procedure or Function
'              does.  If it is too long, consider decomposition.
' Parameters:  List of input parameters.
' Output:      The relevance of the returned value.  Only
'              include if this is a Function
```

2 Indentation

Four spaces should be used as the unit of indentation. This avoids excessive horizontal spread across the screen in deep sections of source code.

3 Comments

Comments should ***not*** be enclosed in large boxes drawn with asterisks or any other characters. Also, consider that many people believe that frequency of comments sometimes reflects poor quality of code. If you are about to add a comment, take a moment to see if you can rewrite the code to make it clearer.

3.1 Single Line Comments

These should be indented to the same level as the code it is referring to and be preceded by a single blank line.

```
If Condition Then

    ‘ This is a single line comment
End If
```

3.2 Trailing Comments

These can be located on the same line as the code they are describing. However, they must be short and should be shifted to the far right. If there are multiple trailing comments in a given method, they should be aligned with one another. The use of this ‘ comment delimiter to comment out chunks of code is acceptable because of the ease of un-commenting individual lines at a later date:

```
If foo > 1 Then
‘    int i = 0
‘    i = i + 1
‘    foo = i
    Exit Sub          ‘ Explain why here
End If
```

3.3 Comment Format

To allow for easy determination of who has altered pieces of code, and to ascertain when the changes were made, the following format should be adopted for all comments:

```
‘ XYZ – The following will do something new – DD/MM/YY
...

‘ XYZ – DD/MM/YY
‘ This needed some extra explanation...
‘ ...and that is that.
’
```

Where XYZ are the initials of the programmer who has added the comment and DD/MM/YY is the current date/month/year.

4 Declarations

4.1 Option Explicit

The `option explicit` statement must be used at all times. This means that all variables must be declared and not just introduced when the need arises. Using this command tends towards neater and more readable code, especially if trailing comments are appended to declarations:

```
Dim aVariable As SomeType     ‘ This will store X
```

4.2 Initialisation

Where possible all variables should be initialised immediately after being declared. The only time that this can not be done is when some computation is required before the initial value of the variable is known.

4.3 Placement

Declarations should only appear at the start of a block (or clause) of. You should not wait until their first use to declare a variable. The initialisation of variables should ideally appear immediately after the declarations and a single blank line:

```
Function aFunction() As Integer
    Dim int1 As Integer

    int1 = 0
    ...

        If condition Then
            Dim int2 As Integer

            int2 = 0
            ...
        End If
End Function
```

If variable `foo` is still in scope, new variables should not be named using this same name, which would hide the declaration of `foo` at the higher level.

4.4 `Let` Command

The Let command should never be used. There is no benefit from using it for assignments and it adds clutter to the code:

```
Function aFunction() As Integer
    Dim int1 As Integer

    Let int1 = 0              ‘ Should NEVER be used
    int1 = 0                  ‘ Should ALWAYS be used
    ...
End Function
```

5 Statements

5.1 Function Return Values

Because in Visual Basic return values are handled by assigning a value to a pseudo-variable with the same name as the Function, it removes the possibility of referencing the currently assigned return value from within the body of the Function. To overcome this problem, each Function should declare a variable called Result and initialise it immediately. Before completing, the Function should simply provide the variable Result as the returning value:

```
Function aFunction() As Integer
    Dim Result As Integer

    Result = 0                ‘ Our default return value

    ...
    aFunction = Result
End Function
```

5.2 If Statements

The normal case should be presented after the `If`, even if this requires the use of a `Not`. If the condition appears to be large or complex, consider simplification using a boolean function call. Also, consider using a `Select Case` statement if you are coding chains of `If` statements. General `If` statements should be structured in the following way:

```
If Condition Then
    Statements
End If

If Condition Then
    Statements
Else
    Statements
End If

If Condition Then
    Statements
Else If Condition
    Statements
Else
    Statements
End If
```

5.3 Select Case Statements

Start with the most common cases as the top of the code. If there exists a legitimate default case, then including it in the `Case Else` section. Otherwise, use `the Case Else` as a means of displaying an error:

```
Select Case score
    Case Is < 50:      grade = "F"

    Case Is < 100:     grade = "A"

    Case Else:         ' Can't get over 100%!!!!
End Select
```

5.4 For & Do Statements

Limit the nesting of these statements to a maximum of three levels. In the case of For statements, the loop variable should always be included in the Next statement as it clarifies the end points of each loop:

```
For i = 1 To 10 Do
    ...
Next i
```

5.5 Goto & Gosub Statements

The use of the `Goto` statement should only be used for error handling (see *Section 5.8 below*) or exiting a Procedure or Function early and cleaning up. If you require the use of a `Goto` to exit a complex nested structure, then you need to decompose the code. Also, the use of the `Gosub` statement is similarly frowned upon, although there are rare situations when the use of the `Gosub` is acceptable.

5.6 `Exit Sub` & `Exit Function` Statements

The use of the `Exit Sub` and `Exit Function` statements are acceptable in situations where you are certain that no cleaning up is required from your Procedure or Function. However, if there needs to be some level of cleaning up the `Goto` statement must (unfortunately) be used:

```
Sub doSomething()
    ...
    If Not canProceed() Then
        Exit Sub              ‘ No cleaning up needed
    End If
    ...
End Sub

Sub doSomething()
    ...
    If Not canProceed() Then
        Goto doSomething_Exit  ‘ We require cleaning up
    End If
    ...
doSomething_Exit:
    ‘ Clean up here
End Sub
```

5.7 `Exit Do` & `Exit For` Statements

Although these statements can make it difficult to follow code, their use is sometimes economical. For instance, the following section of code should never be produced, but an `Exit For` statement used:

```
For index = lbound(items) to uBound(items)
    If items(index) = searchValue Then
        found = True
        foundIndex = index
    End If
Next index

If found Then . . .
```

5.8 Error Handling

Unless absolutely certain that a Procedure of Function does not need an error handler, the following format should be implemented:

```
Sub doSomething()
    On Error Goto doSomething_Error
    ...

doSomething_Exit:
    Exit Sub
doSomething_Error:
    ‘ Handle the error
End Sub
```

Notice that the two labels that distinguish between an error and the normal end to the Procedure are the Procedure name appended with either _Exit or _Error. This method of error handling is fine for simple error checking. However, when you require different error handling dependant upon the location within the code where the error was detected, Visual Basic does not make it easy to implement this. One solution is to set flag variables to indicate where in the code you are. A much preferred method is to temporarily disable error trapping:

```
Sub doSomething()
    On Error Goto doSomething_Error
    ...

    On Error Resume Next
        DoSomethingSpecial
        If Err Then
            ' Handle the special error
        End If

        On Error Goto doSomething_Error       ' Return to normal error
                                              ' trapping

    ...
doSomething_Exit:
    Exit Sub
doSomething_Error:
   ' Handle the normal error
End Sub
```

6 White Space

6.1 Blank Spaces

Blank spaces should be used in the following circumstances:

- A blank space should appear after commas in argument lists.
- A binary operator should be separated from its operands with a blank space.

6.2 Blank Lines

Blank lines should be used in the following circumstances:

- Between the local variables in a procedure and the initialisation of them.
- Between the initialisation of the local variables in a procedure and the first line of code.
- Before a single line comment.
- Between logical sections within a procedure that will increase readability.

7 Naming Conventions

To overcome the problems of Visual Basic being very weakly typed, the naming conventions can be quite complex. However, it is beneficial to encode into the variable names some information that can prevent simple mistakes and to highlight the use of the variable.

7.1 Variables

The standard variable name should consist of a normal name for the variable preceded by two identifiers. The first should be one of the following:

Initial Identifier	Description
g	This denotes that the variable has global scope. That is, it was declared with the `global` keyword.
m	This denotes that the variable has Module (or Form) scope. That is, it was declared in the `general declarations` of a Module or Form.

	The *absence* of a letter denotes that the variable has local scope. That is, it was declared in a Procedure or Function.

The second letter encodes the type of the variable according to the following table:

Second Identifier	Type	More Details
i	Integer	16-bit signed integer
l	Long	32-bit signed integer
s	String	A VB string
n	Numeric	Integer of no specified size (16- or 32-bit)
c	Currency	64-bit integer scaled by 10^{-4}
v	Variant	A VB variant
b	Boolean	A native boolean data type
dbl	Double	A double-precision floating point number
sng	Single	A single-precision floating point number
flt	Float	A floating point number of no specified precision
byte	Byte	An 8-bity binary value
obj	Object	A generic object variable
ctl	Control	A generic control variable

The actual name of a variable should be nouns in mixed case with the first letter of each internal word being a capital. Do not use all capitals for acronyms. Names should be designed to indicate its intended use to a casual observer. Some examples are: `gsPersonName`, `mdblInterestRate`, `nLoopCounter`.

7.1.1 User Defined Types

When a user defined type is implemented, the name of the Type should be preceded by a single letter `T`:

```
Type TEmployee
    NId      As Long
    sSurname As String
    cSalary  As Currency
End Type
```

7.1.2 Arrays

Arrays should use the same naming convention as for regular variables. The only difference is that arrays should be named as plurals.

7.2 Procedures

Procedure names should be verbs in mixed case with the first letter being lowercase, and the first letter of each internal word being a capital. Examples include `run`, `findNextCustomer`, `editHtmlSource`.

7.3 Constants

Constants should be all uppercase with internal words separated with the _ underscore. They should also be declared with the `Global` keyword:

```
Global Const MIN_WIDTH = 0;
Global Const MAX_WIDTH = 1000;
```

7.4 Controls

All controls on a form should be renamed immediately they have been placed. This is because if you write code in an event procedure for a control and then rename it an orphan control procedure is created. For all labels that are only used for static messages (e.g. labeling a Text Box as being for the name field), a control array should be created with the name lblPrompt. The reasoning for using a control array is that it will only take one name up in the form's name table.

For all other controls, the name should be prefixed with a code representing the type of control that it is. These prefixes are shown below:

Prefix	Control Type
cbo	Combo Box
chk	Checkbox
cmd	Command Button
dat	Data Control
dir	Directory List Box
dlg	Common Dialog Control
drv	Drive List Box
ela	Elastic
fil	File List Box
fra	Frame
frm	Form
gau	Gauge
gra	Graph
img	Image
lbl	Label
lin	Line
lst	List Box
mci	MCI Control
mnu	Menu Control
mpm	MAPI Message
mps	MAPI Session
ole	OLE Control
opt	Option Button
out	Outline Control
pic	Picture
pnl	Panel
rpt	Report
sbr	Scroll Bar (There is no need to distinguish orientation)
shp	Shape
spn	Spin
ssh	Spreadsheet Control
tgd	Truegrid
tmr	Timer
txt	Text Box

If there is a concept called the CustomerName, then that is what is should be called everywhere (e.g. NOT CustName, CustLabel etc.). The variable would be called `sCustomerName`, the Text Box that allows you to enter it would be `txtCustomerName` and a Combo Box of all customer names would be called `cboCustomerName`.

7.4.1 Menu Controls

Menu Controls require a quick addition to the current naming convention. The generally accepted name for a Menu Control is the complete path down the menu structure. This has many benefits including the encouragement of creating fairly shallow menu structures (Who wants `mnuFileExportToFileOnNetwork` to be the name of a Control?). Examples are `mnuFileNew`, `mnuEditCopy`, `mnuInsertFootnote`.

7.5 Objects

There are a certain number of objects that are used very frequently, and these should be prefixed with the following characters:

Prefix	Object
db	Database
ws	Workspace
rs	Recordset
ds	Dynaset
ss	Snapshot
tbl	Table
qry	Query
tdf	TableDef
qdf	QueryDef
rpt	Report
idx	Index
fld	Field
xl	Excel Object
wrd	Word Object