

## Best practices for VBA coding

This article will focus on the idea of establishing some good coding practices for writing standard VBA code.

### Best practices for VBA coding

There is always some variation in opinion about what the best coding practices are. You will want to discuss the topic in your organization before determining exactly what your best coding practices for writing VBA code will be. Here are some common suggestions that will help to provide a starting point for you.

#### 1) Establish a naming convention for entities and variables

The aim of Naming Conventions is to improve readability and provide a consistently applied set of rules for naming variables, methods, classes, and other entities native to a particular programming language / environment.

The first most important and general rule of thumb is that a name be meaningful; it should reflect what the entity does (for methods, procedures, functions), or represents (for variables, parameters, classes, and objects). Avoid meaningless variable names, for example use `ilIndex` rather than `i` for an iterator variable since it better shows what the variable is being used for. Using mixed case can also help to separate the parts of a name and make them easier to recognize.

#### 2) Variable Initialization and use

Initializing a variable shortly after it is declared helps to ensure that it always has a known value and reduces the likelihood of the code failing unexpectedly. Choose a value that makes sense, given what the variable will be used for. Avoid reusing the variable for something else later in the code since this makes the code more difficult to interpret for other developers. For example, say our program declares an iterator variable called `ilIndex` and sets it to `-1` shortly after it is declared. We expect the variable to be set to the number of items in our list later in the program. If the variable is still set to `-1` when it comes time to iterate through our list then we know that our code is incorrect and needs updating since no list will have fewer than zero elements.

#### 3) Variable Declaration

It is advisable to declare all variables which you use in your VBA code and to enforce this using the 'option explicit' module level tag. This avoids problems where a misspelled variable causes the implicit creation of a new variable which then ends up with an unintended value resulting in unexpected results.

#### 4) Function (or method) arguments

All arguments to public functions should be verified for validity before they are used. Public functions are exposed to other modules, classes or worksheets which may be developed by someone else who is less familiar with what your intentions were when you wrote the routine. Private functions are less likely to be used by another programmer.

#### 5) Enumerated types

Use enumerated types rather than so called 'magic numbers' for variables which are expected to have a particular set of values. This greatly reduces the chance of a variable being assigned an inappropriate value. Examples include days of the week, months in the year or any other finite set of values whose elements can be represented by constants.

**6) Style and formatting**

Using a consistent style for indenting and commenting code makes it easier for developers in your organization to read and understand each others code. There are lots of options to consider here and you really need to pick rules which fit your organization. The online articles listed at the end of this article go into this topic in more detail if you are interested in developing programming style rules for your organization.

**7) Avoid 'use' or at least 'overuse' of the GoTo statement**

Adding too many 'GoTo' statements into Visual Basic code can make it difficult to understand. It is reasonable to use a 'GoTo' jump within a routine for error handling purposes but it is generally not a good idea to have jumps for other purposes since it makes the code hard to read.

**8) Use VB constants**

When working with Microsoft objects from VBA code it is recommended that you use VB constants rather than simple integers or text when setting the properties. This is another way to reduce the chances of assigning an inappropriate value to a property. An example is the vbOK constant which is returned by a MsgBox call when the OK button is pressed. Checking that the result is equal to vbOK is safer then checking that the result is equal to one and it is also easier to read.

**9) Error handling**

The "On Error Resume Next" command should never be used as your primary error handling mechanism since this can hide problems in your code base. It is better to use "On Error GoTo" and jump to an error handling code snippet at the bottom of your routine. If the situation is one that you can recover from then you can always use the "Resume Next" command inside your error handling code to do the equivalent when you know it is safe. If you choose to exit the routine within your error handling code then you need to ensure that you undo anything that has been partially completed when the error occurred so that you do not create problems elsewhere in your program. Check to see if you need to close a file, reset the calculation mode or call an unload method on an instance of an object before you leave.

In addition to the items mentioned above you may also want to develop some methods for standardizing inputs that come from worksheets. Often it is easier to work with Visual Basic arrays, rather than the Range data structure which may be passed to a function from a spreadsheet call. Having standard utility routines which convert a Range into a Visual Basic array or that extract the row or column attributes from a Range can be useful in preventing simple coding errors that might come up if you have a lot of duplicate code which does this sort of thing.

**Conclusion**

Using good coding practices when developing any type of code, including prototype code, will help you to achieve better, more consistent and easier to understand results.

**References**

1. Stephen Bullen, Rob Bovey, John Green 'Professional Excel Development: The Definitive Guide to Developing Applications Using Microsoft® Excel and VBA®' - Addison Wesley Professional – Published Feb 1, 2005
2. 'Microsoft Visual Basic 6.0 Programmer's Guide' – Microsoft Press – Published 1998
3. Steve McConnell 'Code Complete, Second Edition' – Microsoft Press – Published 2004
4. B. W. Kernighan and P. J. Plauger, The Elements of Programming Style 2nd Edition, McGraw Hill, New York New York, 1978.

**OnLine References**

[http://en.wikipedia.org/wiki/Programming\\_style](http://en.wikipedia.org/wiki/Programming_style)

[http://en.wikipedia.org/wiki/Coding\\_conventions](http://en.wikipedia.org/wiki/Coding_conventions)

**Disclaimer**

Your use of the information in this article is at your own risk. The information in this article is provided on an "as is" basis and without any representation, obligation, or warranty from FINCAD of any kind, whether express or implied. We hope that such information will assist you, but it should not be used or relied upon as a substitute for your own independent research.

Copyright © 2007 FinancialCAD Corporation. All rights reserved. FinancialCAD® and FINCAD® are registered trademarks of FinancialCAD Corporation. Other trademarks are the property of their respective holders. This email is for informational purposes only. FinancialCAD Corporation MAKES NO WARRANTIES, EXPRESSED OR IMPLIED, IN THIS SUMMARY.