# Optimizing VBA Code

## Part One: Variables

by Robert Delwood

This is the first in a series of articles about optimizing VBA (Visual Basic for Applications) code. The information here is drawn from my book, The Secret Life of Word: A Professional Writer's Guide to Microsoft Word Automation.

VBA is known to be a slow language. Often this lack of speed is not a problem. If the routines are small enough or if the data being processed is not too large, the difference is not noticeable. However, as you gain experience with VBA and create longer and more complex programs, the suggestions described in this series can help improve the performance of your code.

## Declare all variables

VBA does not require variables to be declared (with a `Dim` statement) prior to use. That means you can introduce variables at any point, with no declaration. This is convenient, but can be inefficient. VBA treats variables that hold string or text values differently from those holding numbers, and within numbers, VBA treats integers differently from real numbers. However, if you do not declare the type of a variable, it will be created with the type Variant, which is a versatile, but inefficient and slow data type.

As a matter of best practice, always explicitly declare variables. Here is a guide to the most common data types:

- Strings or text should be declared as String.
- Integers should be declared as Long or Integer.
- Numbers requiring decimal places should be declared as Double or Single.
- Items that are objects (rather than strings or numbers) should be declared as the exact type they are, and not as the generic Object type.

For Word programming, you often need to know the exact type of an object, such as Paragraph, Hyperlink, or Range. IntelliSense (the IDE's word completion feature) helps you select the correct type. You can use Word's document object model help to select the Word object type.

Declaring the data type allows Word to use "early binding." Early binding lets Word use declared variables considerably faster than undeclared variables, since it doesn't have to convert the variable each time it encounters it. It also allows you to use IntelliSense in the IDE.

## Give each variable its own As statement

Each variable should have its own `As` statement, since `As` actually implements the type. For example, consider the follow line:

```
Dim a, b, c as Long
```

Many users think that `a`, `b`, and `c` are all declared as Long. Actually, only the last one, `c`, is. The variables `a`, and `b` are implicitly declared as Variant since neither has an `As` statement associated with it. What you should do is either:

```
Dim a as Long, b as Long, c as Long
```

Or

```
Dim a as Long
Dim b as Long
Dim c as Long
```

## Use Option Explicit

The `Option explicit` statement requires that each variable must be explicitly declared prior to use. Use this statement once at the start of each code module (`UserForm`, `Module`, and `Class` Module).

To automatically have the statement included in any new module, do the following: from any Word document, click **ALT-F11** to get into the IDE, then Tools|Options, Editor tab, and in the Code Settings panel check Require Variable Declaration.

In addition, `Option Explicit` helps prevent misspellings and possible debugging problems. For example, in the following code snippet, one variable is misspelled.

```
Dim isCorrectName as Long
isCorrectName = isCorrectName + 1
if ifCorrectName > 0 then msgbox isCorrectName & " name has been located."
```

Without `Option Explicit`, any variable not previously encountered will be instantiated as a legitimate variable. In the example, without `Option Explicit`, the misspelled variable `ifCorrectName` will be treated as a legitimate variable, and the `if` statement will never be `true`. The variable, which should be `isCorrectName` but is misspelled as `ifCorrectName`, will never be incremented and will stay at the default value of zero.

Using `Option Explicit`, the misspelled variable, `ifCorrectName` will cause a compile-time error, which can be easily fixed.

## Summary

Using just a few simple techniques can help you ensure that your variables will be declared and used in the most efficient manner possible. In the next part of this article, I will discuss how to improve the performance of loops in VBA.