VB6 Subroutines and Functions

Malcolm Moore

VB6 - Subroutines

When developing a program it is quite common to realise that the same (or a similar) block of software is used in several places, and it becomes far more expedient (and accurate) to create a Subroutine that performs the repetitive process as a branch off the main program. By constructing a program in this fashion debugging can be considerably simplified, because that one piece of programming works in many places, making it much less prone to typing and transfer errors while developing the program.

There are fundamentally two types of Subroutines – Public and Private, and the difference is that a Public Subroutine can be 'seen' from anywhere in a VB Project; whereas a Private Subroutine can only be seen from within that Form or Module in a VB Project. To keep programming simple, it is best to aim to have all the Subroutines classified as Private, so that are only seen inside their Form or Module.

When you set up a Form and include Command Buttons (and every other visual Object), every one of these visual Objects has a Private Subroutine attached to the Object so that when the Object is alerted (for example by a Mouse 'click' or 'double-click' etc) then the Subroutine is initiated and the programming that is put in this subroutine causes the Project to respond.

For example, here is a small Subroutine where when this Command Button is clicked, only this Subroutine responds, and calls another Subroutine (located in a Form called frmRecent1) to change some data and then update the screen by yet another Subroutine.

```
Private Sub cmdLastSecurity_Click()
  'Move up the Recent List, and show the previous graph
  frmRecent1.RecentStep (-1)
  UpdateScreen
End Sub
```

The first line tells us that the subroutine is called cmdLastSecurity, and that it responds to a Mouse Click over that Command Buttons' area, and no data is being carried into the subroutine.

The Comment line (in Green) tells us what the Subroutine will do, and this line is vitally important as it is common to come back to programs several years after they have been created to make a change; and having a few lines of comments saves hours of frustration.

The next line tells us that the Subroutine called RecentStep is called and –1 is carried into this Subroutine as the first variable. Obviously several variables can be carried into the called Subroutine, separated by commas.

The next line tells us that the screen will be updated by a local Subroutine called UpdateScreen, and nothing is being carried into that Subroutine, so it does not have brackets after its called name.

When these action have been sequentially completed, this cmdLastSecurity Subroutine goes back to sleep until it is next activated by the Mouse click over it.

As it turns out both RecentStep() and UpdateScreen() are both rather large and complex Subroutines, and RecentStep() is a Public Subroutine because it is seen from other forms in this particular Project, and the start of the Subroutine is as follows:

```
Public Sub RecentStep(Movement As Integer)
  'Move the counter to bring up another Security Symbol
  Dim lRecCount As Integer      'The Index for the highlighted Security
  'About 50 lines of code here
End Sub
```

So now we can see that the –1 is transferred into this Subroutine as an Integer called Movement, and this integer affects the counters' position to call up the appropriate security symbol for when the screen is to be updated

VB6 - Functions

The difference between a Subroutine and a Function is that although both a Subroutine and a Function accepts one or more values into it when either is called up, the Function then returns a value back into the variable on that parent's line where the program branched out to the Function.

Public Function

A Public Function is a bound routine that uses Global variables (that are common and defined near the top of the program module). This takes a program from a long scroll of command lines into a number of blocks that can call each other as required. Hidden is the fact that by having the program broken into functional blocks, new features and operations can be created simply created either by calling a choice of these blocks or by adding another functional and using some of the existing functional blocks.

Private Function

A Private Function is a bound routine that has variables assigned to that function within the boundaries of the Function, and not beyond. This type of Function is very useful for performing a highly repetitive calculation, as it can accept variables being passed into it, and it responds with a variable being passed out of it – ***as the name of the function***.

For example if we were to make up a Private Function for a the hypotenuse of a right-angled triangle with the formula:

```
Hypotenuse = Sqrt(Side1^2 +Side2^2)
```

We need to include two variables (Side1 and Side2) inside the Private Function, and both these are single precision variables, as is the result of the Private Function. In this case, this Private Function can be written as:

```
Private Function Hypotenuse (Side1 As Single, Side2 As Single) As Single
Hypotenuse = Sqrt(Side1^2 +Side2^2)
End Function
```

As this is a mini program the (private) variables are all defined in the top line then the maths worked out and the variable 'Hypotenuse' is given the answer, then on ending the private function, that variable can then pass out the resultant value.

To use this Private Function in a larger program – which will have different variables referring to this Private Function – we need to pass the variables into the function and pass the result out of the function. Here is an example of this in use in a building program:

```
RafterLength = Hypotenuse (RoofPitch, HalfSpan)
```

Elsewhere in the module, the RafterLength, RoofPitch and HalfSpan would have been dimensioned (Dim) as single precision variables, and in this case the current values of the two variables in brackets would have been passed into the Private Function, and the structure of the equation will pass the result of the Private Function into the variable 'RafterLength' on the left-hand side of the equation.

Private Functions can be recursive, that is, pass out a result that will then change one (or more) of the original variables being passed into that Private Function.