Scope Of Variables And Procedures

This page describes what it meant by the *Scope* of a variable or procedure.

Introduction

The term *Scope* is used to describe how a variable may be accessed and how a procedure may be called. Depending on where and how a variable is declared, it may be accessible only to a single procedure, to all procedures within a module, and so on up the hierarchy of a project or group of related projects. The term *visibilty* is also used to describe *scope*. The two terms are equivalent. There are four "levels" of scope.

- Procedure Scope
- Module Scope
- Project Scope
- Global Scope

Procedure Scope

Procedure scope means that a variable is accessible (it can be read and modified) only from within the procedure in which it is declared. For example,

```
Sub TestSub()
    Dim X As Long
    Dim Y As Long

    X = 1234
    Y = 4321
    Debug.Print "X: "  & X, "Y: " & Y
End Sub
```

In this code, the variables `X` and `Y` can be accessed only from code within the `TestSub` procedure. These variables are created when `TestSub` is called and they are destroyed when `TestSub` ends. You cannot modify these variables from any other procedure. Procedure scope trumps other scope levels, so if you have, in addition to these variables, variables with the same name declared at the module level (see *Module Scope* below), the code within `TestSub` uses the varibles declared within the procedure, not the variables with the same name at the module level. (See *Resolving Scope Conflicts* below.)

Module Scope

Module scope means that a variable can be declared before and outside of any procedure in a regular code module. If you use `Private` or `Dim` to declare the variable, only procedures that are in the same module can access that variable. Since a module level variable is not part of any procedure, it will retain its value even after the procedure that changes its value has terminated. For example,

```
Dim ModVar As Long
```

The variable `ModVar` can be read or changed from any procedure in the same module as the declaration. However, it cannot be accessed from a procedure that resides in another module. Separate modules may declare variables with the same name (e.g., both Module1 and Module2 both can have a module-level variable named `ModVar`). Each module-level variable will be accessed by procedures within the same module. If both Module1 and Module2 have a module-level variable named `ModVar`, procedures within Module1 will access the variable `ModVar` defined within Module1 and procedures within Module2 will access the variable `ModVar` defined in Module2. Variables with the same name in separate modules are completely independent of one another, even though they have the same name.

You can use either `Dim` or `Private` to declare a module level variable that is accessible only within that module. That is, the following declarations are functionally equivalent.

```
Dim ModVar As Long
Private ModVar As Long
```

Project Scope

Project scope variables are those declared using the `Public` keyword. These variables are accessible from any procedure in any module in the project. In Excel, a *Project* is all of the code modules, userforms, class modules, and object modules (e.g,. `ThisWorkbook` and `Sheet1`) that are contained within a workbook. You can't directly declare a variable that has Project scope but not Global scope (see *Global Scope* below). In order to make a variable accessible from anywhere in the project, you must use the `Public` keyword in the declaration of the variable. However, this makes the variable accessible to any other project that reference the project containing the variable. If you want a variable to be accessible from anywhere within the project, but not accessible from another project, you need to use `Option Private Module` as the first line in the module (above and outside of any variable declaration or procedure). This option makes *everything* in the module accessible only from within the project. This means also that if you *do* want some project variables to be accessible from other projects and other project variables to be inaccessible from other projects, you need to declare them in separate modules. The project variables that should *not* be accessible to other projects should be declared in a module that has the `Option Private Module` directive. Variables that should be accessible to other project should be declared in a different module that does not use the `Option Private Module` directive. In both cases, however, you need to use the `Public` keyword.

Global Scope

Global scope variables are those that are accessible from anywhere in the project that declares them as well as any other project that references the first project. To declare a variable with global scope, you need to declare it using the `Public` keyword in a module that does *not* use the `Option Private Module` directive. In order to access variables in another project,

you can simply use the variable's name. If, however, it is possible that the the calling project also has a variable by the same name, you need to prefix the variable name with the project name. For example, if Project1 declares a global variable named `SomeVar`, and Project2 references Project1, code that is in Project2 can access `SomeVar` with either of the following lines of code:

```
SomeVar = 1234
Project1.SomeVar = 1234
```

If both Project1 and Project2 have variables with at least project scope, you need to include the project name with the variable. For clarity and maintainability, you should *always* include the project name when accessing a variable that is declared in another project. Even if this is not necessary, it makes the code more readable and maintainable.

There is no way to give some variables project, but not global, scope and give others in the same module global scope. Project versus global scope is handled only at the module level, not at the variable level.

Scope And Procedures

Like variables, procedures (`Sub`, `Function`, and `Property` code) also follow the rules of scope, but differ somewhat in the implementation. Any procedure declared in any module (here, we are referring only to code modules, not object or class modules -- see *Scope And Object Modules* below), can be called by any other procedure in the same project, *unless* the procedure uses the `Private` declaration. Using the `Public` keyword in the declaration is the same as omitting the scope declaration entirely. For example, the following procedures may be called from any other procedure in any other module of the project.

```
Public Sub TheProcName()
    Debug.Print "The Proc Name"
End Sub

Sub TheProcName()
    Debug.Print "The Proc Name"
End Sub
```

To create a procedure that is accessible only from within the module in which it resides, you need to use the `Private` declaration:

```
Private Sub TheSub()
    Debug.Print "abc"
End Sub
```

The `Private` declaration prevents the procedure from being listed in Excel's Macros dialog, but the procedure can still be executed from the Macros dialog by typing in its name.

As was the case with variables, the `Public` keyword makes the procedure accessible to all procedures in the project as well as to procedures in another projects that references the first. To make a procedure accessible from anywhere in the project that contains it, but not from other projects, you can use the `Option Private Module` directive to restrict access only to procedures within the same project. Therefore, if you have some procedures that should be accessible to other projects, put those

procedures in a module that does *not* use the `Option Private Module` directive, and include `Option Private Module` in all other modules. The only way to prevent a procedure from being used by other projects is to use `Option Private Module`.

Resolving Scope Conflicts

When there are variables with the same name but with different scopes, VBA always uses the variable with the least scope. For example, look at the following code in `Module1`:

```
Private XYZ As Long

Sub Proj2AAA()
    Dim XYZ As Long
    XYZ = 1234
    Debug.Print XYZ
End Sub
```

Here, there are two variables named `XYZ`, one with procedure scope and one with module scope. The code within the `Proj2AAA` procedure will access the variable that is declared within that procedure, not the variable declared within the module, since the procedure variable has more immediate scope. If, however, you need to access the module variable `XYZ`, you can prefix the variable name with the module name. For example,

```
Private XYZ As Long     ' module level variable

Sub Proj2AAA()
    Dim XYZ As Long     ' procedure level variable
    XYZ = 1234          ' access procedure variable
    Module1.XYZ = 321   ' access module variable
    Debug.Print "Procedure: " & XYZ & "  Module: " & Module1.XYZ
End Sub
```

It is possible to have `Public` variables with the same name declared in two different modules. For example, both `Module1` and `Module2` can use `Public XYZ As Long`. The code in each module that uses variable `XYZ` will be accessing its own copy of the variable named `XYZ`. The two variables are entirely independent of one another -- they merely have the same name. To access `XYZ` in another module, you can prefix the variable with the module name:

```
Sub AAA()
        Module1.XYZ = 1234
        Debug.Print Module1.XYZ
End Sub
```

As a general rule, though, you should never use project scope variables with the same name. It is much to easy to lose track of which module's variable is being accessed. If you still feel the need to use two variables with the same name, use the `Private` declaration to avoid conflicts. (As an aside, a procedure many not have the same name as any module in the project. Otherwise, you'll get a compiler error.)

Scope And Object Modules

So far, we have looked at variable and procedure declarations in standard code modules. This section looks at object modules. An object module is one of the following: (1) a Class module, (2) the code module for a UserForm, (3) the ThisWorkbook code module, or (4) one of the Sheet modules. Procedures in an object module are `Public` if the `Public` keyword is used or if neither `Public` nor `Private` is used. However, you cannot call a procedure defined in an object module without using the object name. For example,

```
''''''''''''''''''''''
' In Class1 module
''''''''''''''''''''''
Public Sub PubSub()
        Debug.Print "PubSub"
End Sub
Sub UnSub()
        Debug.Print "UnSub"
End Sub

''''''''''''''''''''''
' In Module1 module
''''''''''''''''''''''
Sub AAA()
    Dim C As Class1
    Set C = New Class1
    C.PubSub  ' Legal
    C.UnSub   ' Legal
    PubSub    ' Not Legal, missing the Object name
End Sub
```

As shown above, you must always use the object's name when calling a procedure defined within an object module. Because of this restriction, though, it is perfectly legal and, in some cases desirable, to have `Public` procedures with the same name in different object modules. This is because the call to the procedure always identifies the object containing the procedure.

Instancing Property Of A Class

In VBA, a class module has a property named `Instancing`. (This property can be changed only at design time -- you cannot modify the property value with code at run time.). This property has a value of either `Private` or `PublicNotCreatable`. If a class has `Private` instancing, that class can be used, and a variable declared of that type, only within the project that contains the class module. The class is completely invisible to other project that reference the project containing the class. (Well, not quite invisible. Another project can declare a variable `As Object` and set that to a new instance of the class so long as the class is created in the project the defines it.)

If the instancing property is `PublicNotCreatable`, the class behaves normally when used within the same project, but a variable can be declared of that class type in other projects. The other project cannot *create* a new instance of the class, but can have a variable of the class's type. To allow another project to use a new instance of the class, the project containing the class must provide a global-scope function that creates a new instance of a class and returns it to the caller. For example, suppose Project1 contains a class named *Class1*, whose Instancing property is *PublicNotCreatable*. Suppose also that Project2 references Project1.

```
'''''''''''''''''''''''''''''''
' In Project2
'''''''''''''''''''''''''''''''
Sub AAA()
        Dim C As Project1.Class1      ' Legal
        Set C = New Project1.Class1       ' Illegal -- not allowed tc
End Sub
```

For the code in Project2 to use a new instance of Class1, Project1 must create the instance of the class and return that to the calling code:

```
'''''''''''''''''''''''''''''''
' In Project1
'''''''''''''''''''''''''''''''
Public Function GetClass1() As Class1
        Set GetClass1 = New Class1
End Sub

'''''''''''''''''''''''''''''''
' In Project2
'''''''''''''''''''''''''''''''
Sub AAA()
        Dim C As Project1.Class1   ' Legal
        Set C = Project1.GetClass1()
End Sub
```

The code in Project2 can declare a variable of type `Class1`, but cannot create a new instance of it. Instead, Project1 creates the new instance of `Class1` and returns it to the code in Project2.

The Friend Scope Declaration

In addition to `Public` and `Private`, a procedure in an object module can be declared as `Friend`. A `Friend` procedure may be called only from within the project that contains the class, regardless of the Instancing property of the class. Suppose `Class1` is defined within Project1, and contains the following code:

```
Public Sub PubSub()
    Debug.Print "PubSub"
End Sub

Friend Sub FriendSub()
    Debug.Print "FriendSub"
End Sub

Sub NoSub()
    Debug.Print "NoSub"
End Sub
```

With this code, all three procedures can be called from within Project1, but only `PubSub` and `NoSub` can be called from other projects. `FriendSub` is invisible outside Project1.

This allows you to make only some procedures of a class avaible to other project. This differs from regular code modules, in which global scope is determined on a module by module basis, not on a procedure by procedure basis. You cannot use the `Friend` declaration in regular code modules. It is allowed only in object modules.

This page last updated: 6-January-2008

Created by Chip Pearson at Pearson Software Consulting, LLC
Email: chip@cpearson.com Before emailing me, please read this page.
http://www.cpearson.com/excel/Scope.aspx

Subscribe To The CPearson.com Weekly Excel Newsletter

Main Page | Page Index | Topic Index | What's New
Search
Consulting | About This Site | Downloads | Feedback
Legal & Copyright

About This Web Site — Go To Page
0 - Rate This Page — Submit Rating