Chapter 38: Attributes

Section 38.1: VB_PredeclaredId

Creates a Global Default Instance of a class. The default instance is accessed via the name of the class.

Declaration

```
VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "Class1"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False
Option Explicit

Public Function GiveMeATwo() As Integer
    GiveMeATwo = 2
End Function
```

Call

```
Debug.Print Class1.GiveMeATwo
```

In some ways, this simulates the behavior of static classes in other languages, but unlike other languages, you can still create an instance of the class.

```
Dim cls As Class1
Set cls = New Class1
Debug.Print cls.GiveMeATwo
```

Section 38.2: VB_[Var]UserMemId

`VB_VarUserMemId` (for module-scope variables) and `VB_UserMemId` (for procedures) attributes are used in VBA mostly for two things.

Specifying the default member of a class

A `List` class that would encapsulate a `Collection` would want to have an `Item` property, so the client code can do this:

```
For i = 1 To myList.Count 'VBA Collection Objects are 1-based
    Debug.Print myList.Item(i)
Next
```

But with a `VB_UserMemId` attribute set to 0 on the `Item` property, the client code can do this:

```
For i = 1 To myList.Count 'VBA Collection Objects are 1-based
    Debug.Print myList(i)
Next
```

Only one member can legally have `VB_UserMemId = 0` in any given class. For properties, specify the attribute in the `Get` accessor:

```
Option Explicit
Private internal As New Collection

Public Property Get Count() As Long
    Count = internal.Count
End Property

Public Property Get Item(ByVal index As Long) As Variant
Attribute Item.VB_Description = "Gets or sets the element at the specified index."
Attribute Item.VB_UserMemId = 0
'Gets the element at the specified index.
    Item = internal(index)
End Property

Public Property Let Item(ByVal index As Long, ByVal value As Variant)
'Sets the element at the specified index.
    With internal
        If index = .Count + 1 Then
            .Add item:=value
        ElseIf index = .Count Then
            .Remove index
            .Add item:=value
        ElseIf index < .Count Then
            .Remove index
            .Add item:=value, before:=index
        End If
    End With
End Property
```

Making a class iteratable with a `For Each` loop construct

With the magic value `-4`, the `VB_UserMemId` attribute tells VBA that this member yields an enumerator - which allows the client code to do this:

```
Dim item As Variant
For Each item In myList
    Debug.Print item
Next
```

The easiest way to implement this method is by calling the hidden `[_NewEnum]` property getter on an internal/encapsulated `Collection`; the identifier needs to be enclosed in square brackets because of the leading underscore that makes it an illegal VBA identifier:

```
Public Property Get NewEnum() As IUnknown
Attribute NewEnum.VB_Description = "Gets an enumerator that iterates through the List."
Attribute NewEnum.VB_UserMemId = -4
Attribute NewEnum.VB_MemberFlags = "40" 'would hide the member in VB6. not supported in VBA.
'Gets an enumerator that iterates through the List.
    Set NewEnum = internal.[_NewEnum]
End Property
```

Section 38.3: VB_Exposed

Controls the instancing characteristics of a class.

```
Attribute VB_Exposed = False
```

Makes the class `Private`. It cannot be accessed outside of the current project.

```
Attribute VB_Exposed = True
```

Exposes the class `Public`ly, outside of the project. However, since `VB_Createable` is ignored in VBA, instances of the class can not be created directly. This is equivalent to a the following VB.Net class.

```
Public Class Foo
    Friend Sub New()
    End Sub
End Class
```

In order to get an instance from outside the project, you must expose a factory to create instances. One way of doing this is with a regular `Public` module.

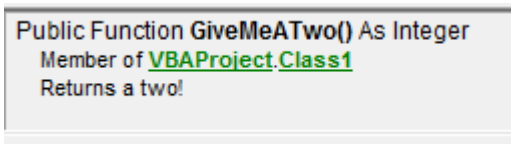
```
Public Function CreateFoo() As Foo
    CreateFoo = New Foo
End Function
```

Since public modules are accessible from other projects, this allows us to create new instances of our `Public` - `Not Createable` classes.

Section 38.4: VB_Description

Adds a text description to a class or module member that becomes visible in the Object Explorer. Ideally, all public members of a public interface / API should have a description.

```
Public Function GiveMeATwo() As Integer
    Attribute GiveMeATwo.VB_Description = "Returns a two!"
    GiveMeATwo = 2
End Property
```



Note: all accessor members of a property (`Get`, `Let`, `Set`) use the same description.

Section 38.5: VB_Name

VB_Name specifies the class or module name.

```
Attribute VB_Name = "Class1"
```

A new instance of this class would be created with

```
Dim myClass As Class1
myClass = new Class1
```

Section 38.6: VB_GlobalNameSpace

In VBA, this attribute is ignored. It was not ported over from VB6.

In VB6, it creates a Default Global Instance of the class (a "shortcut") so that class members can be accessed without using the class name. For example, `DateTime` (as in `DateTime.Now`) is actually part of the `VBA.Conversion`

class.

```
Debug.Print VBA.Conversion.DateTime.Now
Debug.Print DateTime.Now
```

Section 38.7: VB_Createable

This attribute is ignored. It was not ported over from VB6.

In VB6, it was used in combination with the `VB_Exposed` attribute to control accessibility of classes outside of the current project.

```
VB_Exposed=True
VB_Creatable=True
```

Would result in a `Public Class`, that could be accessed from other projects, but this functionality does not exist in VBA.