

VBA Foundations, Part 11

A Tutorial in VBA for Beginners—The Eleventh of a Twelve Part Series

Richard L. Binning / rbinning@attbi.com

Welcome back for another look at VBA Foundations for AutoCAD®. This issue will look at the concept of Events in greater depth. I mentioned in the last issue that events are at the heart of VBA and the AutoCAD® object model has already enabled many objects with the ability to react to specific events. I also told you that with a beginners understanding of events, one can create new events for objects to react to. If you have not been downloading the long version of this series of articles from the PaperSpace downloads page, I would encourage you to do so. We will be looking at even more code in this issue than ever before. Look for the code later in this document for an example using our Purge-n-SaveAll routine that will encompass all the concepts we are learning today. All right! Enough already! Lets get started.

Events are notifications, or messages, that are sent out by whatever application you happen to be working in, AutoCAD® in our example, to inform you about some thing or activity, or to alert you that something has happened or is about to happen. If you look up the definition of an event in the Microsoft Developer Network documentation you will find the following:

An action recognized by an object, such as a mouse click or key press, for which you can define a response. An event can be caused by a user action or a Visual Basic statement, or it can be triggered by the system.

Looking at these descriptions and definitions we see that AutoCAD® itself is anticipating specific actions and providing a mechanism to allow us to add to, modify, or ignore these specific actions. AutoCAD® itself provides three categories of Events:

- **Application Events:** These events are directly related to changes in AutoCAD's state and the application environment. These changes include changes to the AutoCAD® window itself such as the user minimizing or maximizing the window. Additional examples include documents being opened, saved, printed, and closed. Other integrated applications such as an AutoLisp or Arx routine when called, loaded, cancelled, and unloaded are also tracked as Application Level events. *Note: Application Level Events are not automatically loaded when a VBA routine is run. See the following segment for the steps required to create, load, and initialize an Application Level Event.*

To Load or Enable Application Events:

1.) *Insert a class module into your VBA project*

2.) *Give your class a meaningful name such as "myEventClass"*

3.) *Add the following code to your Class:*

```
Public WithEvents App As AcadApplication
```

4.) *Now we need to connect the class to our Application. We can do this from any module. So lets add the following code to the general declarations area of a new module we will insert into our VBA Project. The general declarations area is simply the top of the module.*

```
Dim clsEvents As New myEventClass
```

5.) *Now give your module a meaningful name such as "EventClassLoadingModule" and add the following sub routine as shown below:*

```
Sub InitializeEvents()
```

```
Set clsEvents.App = ThisDrawing.Application
```

VBA Foundations, Part 11

A Tutorial in VBA for Beginners—The Eleventh of a Twelve Part Series

Richard L. Binning / rbinning@attbi.com

End Sub

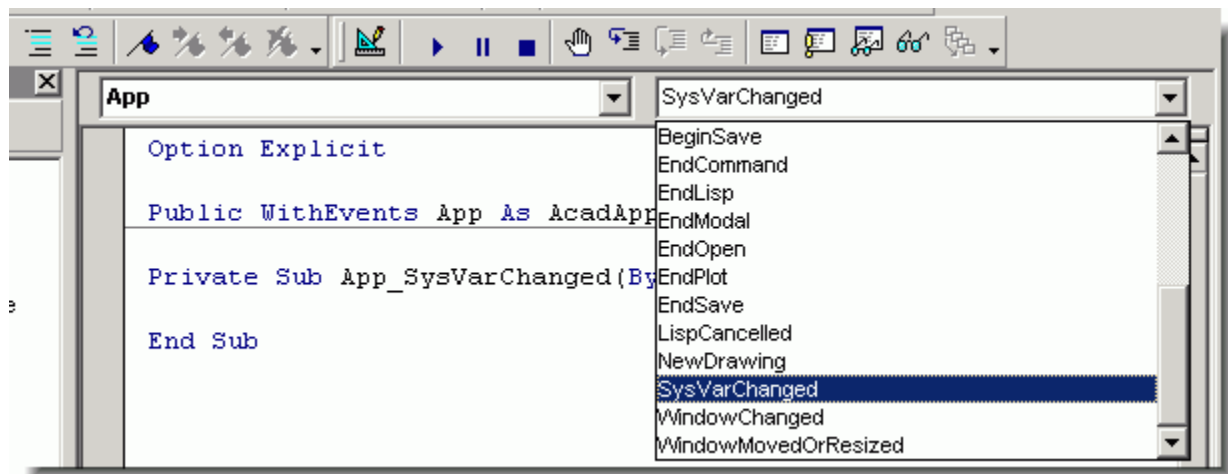
- 6.) *Now to get this class working! Simply call your “InitializeEvents” subroutine. How? Many ways, we can put a “Public” sub in our Module to call it out or in ThisDrawing or on a form or ??? Can you think of other ways?*

```
Public Sub Kick_IT_Off()  
    Call InitializeEvents
```

End Sub

- 7.) *Once this Event controller is initialized, by moving the mouse cursor into the above sub and pressing the “F5” key, you can return to the “myEventClass” class code window and start setting up individual handlers for the event triggers that occur related to the Application itself. Do this by selecting the “App” object in the Object pulldown and then select the desired trigger from the available options in the procedures drop down to the right of the object drop down. This action will result in the creation of skeleton code as shown below (“App_SysVarChanged...”) ready for you to add code to.*

(See graphic Below).



Please be aware that unlike Visual Lisp Reactors, AutoCAD’s VBA enabled application level events are not persistent. In other words, they must be enabled for VBA and any other Applications that might wish to make use of them. Once these application level events have been enabled, you will have a wide range of events available to you. See the chart below for a list of the application level events available and when they are triggered.

VBA Foundations, Part 11

A Tutorial in VBA for Beginners—The Eleventh of a Twelve Part Series

Richard L. Binning / rbinning@attbi.com

AutoCAD Application Level Events			
AppActivate	Triggered just before the main Application window is activated.	EndCommand	Triggered immediately after a command completes.
AppDeactivate	Triggered just before the main Application window is deactivated.	EndLISP	Triggered upon completion of evaluating a LISP expression.
ARXLoaded	Triggered when an ObjectARX application has been loaded.	EndModal	Triggered just after a modal dialog box is dismissed.
ARXUnloaded	Triggered when an ObjectARX application has been unloaded.	EndOpen	Triggered immediately after AutoCAD finishes opening an existing drawing.
BeginCommand	Triggered immediately after a command is issued, but before it completes.	EndPlot	Triggered after a document has been sent to the printer.
BeginFileDrop	Triggered when a file is dropped on the main Application window.	EndSave	Triggered when AutoCAD has finished saving the drawing.
BeginLISP	Triggered immediately after AutoCAD receives a request to evaluate a LISP expression.	LISPCancelled	Triggered when the evaluation of a LISP expression is canceled.
BeginModal	Triggered just before a modal dialog box is displayed.	NewDrawing	Triggered just before a new drawing is created.
BeginOpen	Triggered immediately after AutoCAD receives a request to open an existing drawing.	SysVarChanged	Triggered when the value of a system variable is changed.
BeginPlot	Triggered immediately after AutoCAD receives a request to print a drawing.	WindowChanged	Triggered when there is a change to the Application window.
BeginQuit	Triggered just before an AutoCAD session ends.	WindowMovedOrResized	Triggered just after the Application window has been moved or resized.
BeginSave	Triggered immediately after AutoCAD receives a request to save the drawing.		

- Document Events:** These events are directly tied to the drawings opened by AutoCAD®. Specifically, a document event is triggered when changes occur within or to an AutoCAD® drawing (*.dwg), the objects this drawing contains, or the window it is displayed in. Some of the document level events are duplicates of the application events allowing you, the programmer, additional control and notification down to the specific file. Examples of Overlapping Document Events include: Begin and End events for saving, closing, plotting, and running Lisp routines. Additional Document Level Events include: Document Window changes, Object level changes, and Shortcut menu notifications. *Note: Document Level Events ARE automatically loaded in the “ThisDrawing” module, which is contained in every VBA routine by default. We will not look at enabling Document level events from outside of VBA in this article, but the procedure is well documented in the online help. If still stumped, please post your question to the VBA guild, or email me directly for the answer.*

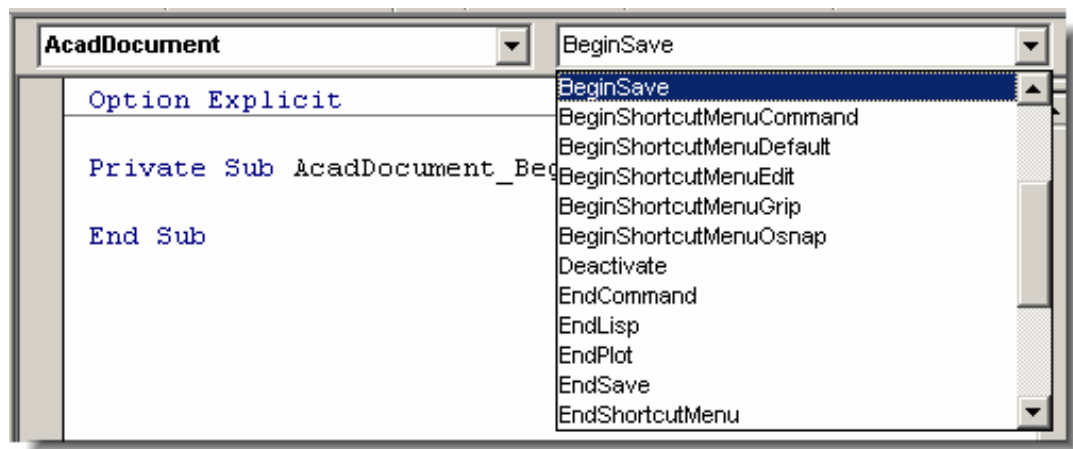
VBA Foundations, Part 11

A Tutorial in VBA for Beginners—The Eleventh of a Twelve Part Series

Richard L. Binning / rbinning@attbi.com

To Enable Document Level Events inside of AutoCAD:

- 1.) Activate the “ThisDrawing” class in the VBA editor by double clicking on it in the project explorer.
- 2.) Select the “AcadDocument” object in the Object pulldown and then select the desired trigger from the available options in the procedures drop down to the right of the object drop down. (See Graphic)



- 3.) This action will result in the creation of skeleton code to be filled in prior to activating.

Option Explicit

```
Private Sub AcadDocument_BeginSave(ByVal FileName As  
String)
```

```
End Sub
```

Did you notice how as soon as you selected the “AcadDocument” object in the left combo box that the skeleton code shown above was inserted into your drawing? Why was the “Begin Save” Event chosen above all others? If we sneak a peak into the Type Library File for AutoCAD we can see that the “Begin Save” event is simply listed as the first choice, even though it is not listed first in the selection box in VBA. (See Graphic Top of Next Page)

Note: Event handlers created using this method apply to the current **active** drawing.

(TypeLibrary View)

VBA Foundations, Part 11

A Tutorial in VBA for Beginners—The Eleventh of a Twelve Part Series

Richard L. Binning / rbinning@attbi.com

AutoCAD Interface Definition

General Information

Library: AutoCAD (AutoCAD 2000 Type Library)
 File: C:\Program Files\ADT33\acad.tlb
 GUID: {C094C1E2-57C6-11D2-85E3-080009A0C626}
 Version: 1.1

_DAcadDocumentEvents {976858E1-0B7B-11D0-89DF-00805FC2FBBA}

Methods

Function BeginSave(ByVal FileName As String) As ???

Triggered immediately after AutoCAD receives a request to save the drawing

Function EndSave(ByVal FileName As String) As ???

Triggered when AutoCAD has finished saving the drawing

Function BeginCommand(ByVal CommandName As String) As ???

(Document Level Events)

AutoCAD Document Level Events			
Activate	Triggered when a Document window is activated.	EndCommand	Triggered immediately after a command completes.
BeginClose	Triggered just before a document is closed.	EndLISP	Triggered upon completion of evaluating a LISP expression.
BeginCommand	Triggered immediately after a command is issued, but before it completes.	EndPlot	Triggered after a document has been sent to the printer.
BeginDoubleClick	Triggered after the user double-clicks on an object in the drawing.	EndSave	Triggered when AutoCAD has finished saving the drawing.
BeginLISP	receives a request to evaluate a LISP expression.	EndShortcutMenu	Triggered after the shortcut menu appears.
BeginPlot	Triggered immediately after AutoCAD receives a request to print a drawing.	LayoutSwitched	Triggered after the user switches to a different layout.
BeginRightClick	Triggered after the user right-clicks on the Drawing window.	LISPCancelled	Triggered when the evaluation of a LISP expression is canceled.
BeginSave	Triggered immediately after AutoCAD receives a request to save the drawing.	ObjectAdded	Triggered when an object has been added to the drawing.
BeginShortcutMenuCommand	Triggered after the user right-clicks on the Drawing window, and before the shortcut menu appears in Command mode.	ObjectErased	Triggered when an object has been erased from the drawing.
BeginShortcutMenuDefault	Drawing window, and before the shortcut menu appears in Default mode.	ObjectModified	Triggered when an object in the drawing has been modified.
BeginShortcutMenuEdit	Drawing window, and before the shortcut menu appears in Edit mode.	SelectionChanged	Triggered when the current pickfirst selection set changes.
BeginShortcutMenuGrip	Drawing window, and before the shortcut menu appears in Grip mode.	WindowChanged	Triggered when there is a change to the Document window.
BeginShortcutMenuOsnap	Drawing window, and before the shortcut menu appears in Osnap mode.	WindowMovedOrResized	Triggered just after the Drawing window has been moved or resized.
Deactivate	Triggered when the Drawing window is deactivated.		

VBA Foundations, Part 11

A Tutorial in VBA for Beginners—The Eleventh of a Twelve Part Series

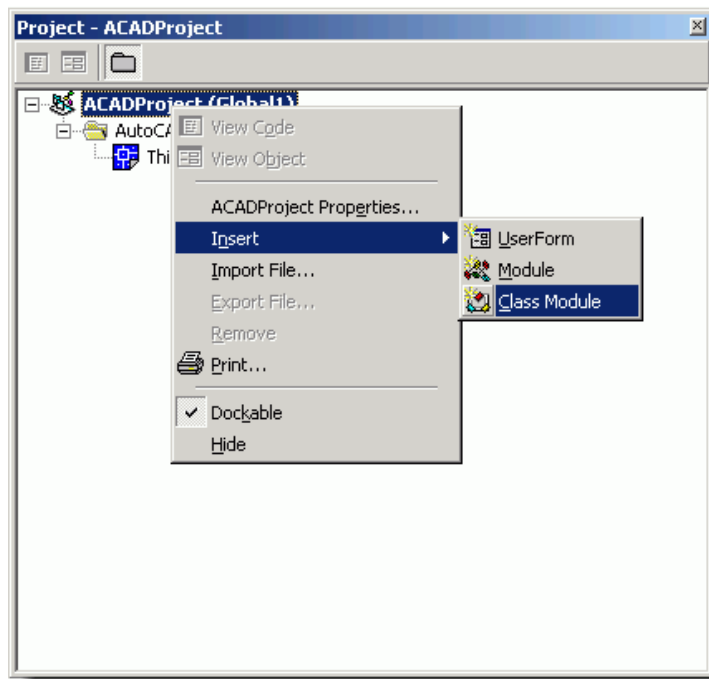
Richard L. Binning / rbinning@attbi.com

- **Object Events:** This type of event is triggered when individual object types are modified. Although, Object Level Events are limited to a single event, **modified**, they can be declared for virtually any of the 40+ types of objects used in AutoCAD® or any of the vertical products. *Note: Object Level Events are not automatically loaded when a VBA routine is run. They can be initialized in the same manner as the Application Level Events.*

To Enable Object Level Events inside of AutoCAD:

Create a class module and declare an object “with events” as a particular type of AcadObject. To actually work this Event, then you must connect to your new class by creating an instance of your Class and creating an instance of the object then setting your class instance equal to the object. Here is how you do that for a Circle object:

- 1.) *Move your mouse to the Project Browser, select your project with the left mouse button and then right click while the mouse is directly over the highlighted project. From the right click menu choose “Insert” and select “Class Module” from the pop up menu shown. (See Graphic Below)*

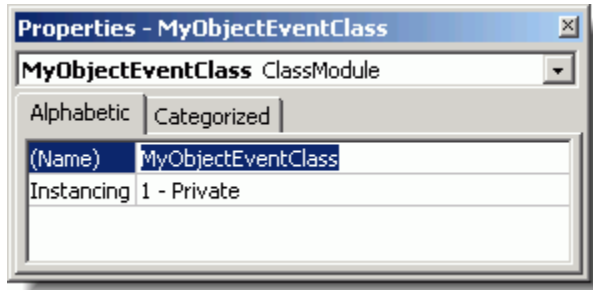


VBA Foundations, Part 11

A Tutorial in VBA for Beginners—The Eleventh of a Twelve Part Series

Richard L. Binning / rbinning@attbi.com

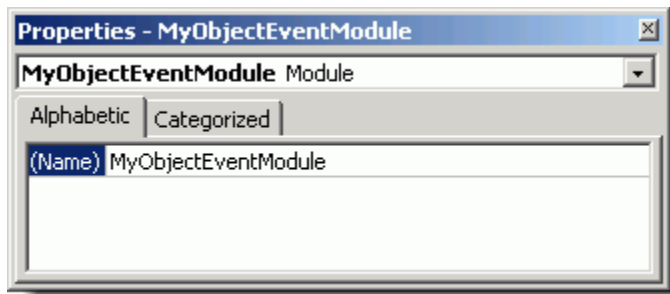
- 2.) *Now select your new class module in the Project Browser so that it is highlighted then move your mouse to the Properties Window and rename your class module to something meaningful such as: MyObjectEventClass. (See Graphic Below)*



- 3.) *With your class module still selected in the Project Browser hit your “F7” function key to switch to the Code Window. Within the code window for the Object Event Class add the following statement:*

```
Option Explicit  
Public WithEvents Object As AcadCircle
```

- 4.) *Now insert a module into your project using the same method as shown in step 1 and 2 above (make sure you select a module instead of a class this time). Now rename your module to something meaningful such as: MyObjectEventModule (See Graphic Below)*



- 5.) *With your Module still selected, switch to the code window and insert the following code into your Module. The first segment goes at the very top of the code window in the area known as the “General Declarations” area.*

```
Option Explicit  
Dim oCircle As New MyObjectEventClass
```

VBA Foundations, Part 11

A Tutorial in VBA for Beginners—The Eleventh of a Twelve Part Series

Richard L. Binning / rbinning@attbi.com

- 6.) *Now insert the following sub procedure in the module:*

```
Sub InitializeEvents()  
  
    'Declare a circle and initialize the necessary properties  
    Dim MyCircle As AcadCircle  
    Dim centerPoint(0 To 2) As Double  
    Dim radius As Double  
  
    'Populate the properties with default values  
    centerPoint(0) = 0#  
    centerPoint(1) = 0#  
    centerPoint(2) = 0#  
    radius = 5#  
  
    'Create the object in the AutoCAD window  
    Set MyCircle = ThisDrawing.ModelSpace.AddCircle(centerPoint, radius)  
    'Connect the new circle to the circle Event Handler  
    Set oCircle.Object = MyCircle  
  
End Sub
```

- 7.) *Now **BEFORE YOU TRY TO RUN THE PRECEDING CODE** switch back to your class module and add the following “Event Handler” for the “Modified” event. (See Graphic Below)*

```
Private Sub Object_Modified(ByVal pObject As IAcadObject)  
    'Note: When coding in VBA, you must provide an event handler for _  
    all objects enabled for the Modified event. If you do not provide _  
    a handler, VBA may terminate unexpectedly. So be sure to run _  
    the Initialize Event in the MyObjectEventModule before this has a chance _  
    to run  
  
    On Error GoTo ERRORHANDLER  
    Dim mytmpCenter As Variant  
    mytmpCenter = pObject.Center  
    MsgBox "Your Object was Modified" & _  
        vbCrLf & "The Center is located at: " & _  
        vbCrLf & mytmpCenter(0) & ", " & _  
        mytmpCenter(1) & ", " & mytmpCenter(2)  
    Exit Sub  
  
ERRORHANDLER:  
  
    MsgBox Err.Description  
  
End Sub
```

Note: You **must** provide an event handler for **all objects enabled** for the Modified event. If you do not provide a handler, VBA may terminate unexpectedly.

VBA Foundations, Part 11

A Tutorial in VBA for Beginners—The Eleventh of a Twelve Part Series

Richard L. Binning / rbinning@attbi.com

- 8.) *Now you can safely run the initialize event to create a circle and connect that particular circle to your event handler. Now switch to AutoCAD's window and zoom in on your new circle that was just added to your drawing. Select the circle and move it.*

Did you see the message box pop up with the new center point? Pretty neat and not a lot of code either?

I have included an event handler for our ever popular Purge-N-Save-N-Close (PSC) macro. Take a look at the code I have included in the following three listings on the next pages.

This code once activated will add right click functionality to our PSC macro and add a Drawing switcher that you may find is a little bit easier than switching using the "Window" pull down method. Watch out for word wrap in the code listings below...a lot of code coming up right now!!!! If you experience problems copying and pasting the code in the listings below, send me an email with "RTCLICK" in the subject and I'll return a working macro file to you!

Listing 1 From a Class Module named: myEventClass

```
Option Explicit

Public WithEvents App As AcadApplication
```

Listing 2 From the Module named: EventClassLoadingModule

```
Option Explicit

Dim clsEvents As New myEventClass

Sub InitializeEvents()
    Set clsEvents.App = ThisDrawing.Application
End Sub

Public Sub Kick_IT_Off()
    Call InitializeEvents
End Sub
```

Listing 3 From the ThisDrawing Module

```
Option Explicit

Public intMnItemCntAdded As Integer
```

VBA Foundations, Part 11

A Tutorial in VBA for Beginners—The Eleventh of a Twelve Part Series

Richard L. Binning / rbinning@attbi.com

```
Private Sub AcadDocument_BeginShortcutMenuDefault _
    (ShortcutMenu As AutoCAD.IAcadPopupMenu)
    On Error Resume Next
    ' Declare a menu item to hold the cursor menu items
    Dim newItem As AcadPopupMenu
    ' Declare a submenuitem to hold dwg names when the value is _
    more than 4 drawings open in the AutoCAD editor
    Dim newSubMenuItem As AcadPopupMenu
    ' Declare a new popupmenu to hold our submenu when needed.
    Dim newMenu As AcadPopupMenu

    If Application.Documents.Count > 1 Then
        'Got more than 1 document
        'Add a string array since we will use it to grab _
        the name of every open drawing for easy file access
        Dim strDocsOpen() As String
        'Add a integer to store the value of the count value
        Dim intCounter As Integer
        Dim oDoc As AcadDocument 'need an object to cycle through all docs
        Dim strCurrDoc As String 'to capture document name
        'Grab Active Document Name
        If Not ThisDrawing.FullName = "" Then
            strCurrDoc = StripPathFromName(ThisDrawing.FullName) 'Get Name
        Else
            'haven't saved this default drawing yet so skip it. _
            Note: this occurs because the .FullName property is not _
            valid until a drawing file is saved.
        End If
        'Ensure our counter is reset every time a rightclick occurs
        intCounter = 0
        'Put our Default MenuItem in place with the code necessary _
        to run the chosen macro
        Set newItem = ShortcutMenu.AddMenuItem _
            (intCounter, Chr(Asc("&")) _
            + "Purge, Save, & Cloze ALL", _
            "-vbarun Close_N_PurgeAllbutCurrent ")
        'Do inline error trapping since we can't insert identical items
    End If
End Sub
```

VBA Foundations, Part 11

A Tutorial in VBA for Beginners—The Eleventh of a Twelve Part Series

Richard L. Binning / rbinning@attbi.com

```
'into the popup menu, we wish to try to insert and skip if an
'error is generated, clearing the error along the way of course.
'increment counter for next item
If Not Err Then
    'no problem with initial menu item so lets add a separator
    intCounter = intCounter + 1
    'check if separator already in place
    If Not ShortcutMenu.Item(intCounter).Type = acMenuSeparator Then
        Set newItem = ShortcutMenu.AddSeparator(intCounter)
    End If
Else
    Err.Clear
End If
On Error GoTo 0 'make sure error is cleared

On Error Resume Next
'reset counter for next loop and reuse the counter variable _
for a new purpose
intCounter = 0
For Each oDoc In Application.Documents
    'Check for "Drawing#" indicating an unsaved default
    'drawing...since FullName prop is not set until save event
    If Not oDoc.FullName = "" Then
        'Check current doc name against first selection
        If Not oDoc.Name = strCurrDoc Then
            'Redimension Array each time through to add 1 new
            'slot per drawing name...this method is not recommended
            'for very large arrays, but should work well for our
            'situation
            ReDim Preserve strDocsOpen(intCounter) 'make our array grow dynamically
            strDocsOpen(intCounter) = oDoc.FullName
            'increment counter for next item
            intCounter = intCounter + 1
        End If 'match name
    End If 'Check default
Next oDoc 'Go get the next one!
```

VBA Foundations, Part 11

A Tutorial in VBA for Beginners—The Eleventh of a Twelve Part Series

Richard L. Binning / rbinning@attbi.com

```
If Not IsEmpty(strDocsOpen) Then 'check for empty array
    'Array found so lets cycle through all elements of array
    'cycle from low index to high index value
    For intCounter = LBound(strDocsOpen) To UBound(strDocsOpen)
        'try to speed things up by not repeatedly calling array element, _
        instead set a variable to hold the name each time through the loop
        strCurrDoc = StripPathFromName(strDocsOpen(intCounter))
        'Add Loop to submenu > 4 drawings open
        If intCounter > 3 Then
            Select Case intCounter
            Case 4
                'Since we have 5 drawings add a submenu to hold additional _
                drawings
                Set newMenu = ShortcutMenu.AddSubMenu(intCounter + 2, "More...")
                If Err Then
                    'must already exist in popup, skipping now
                    Err.Clear
                Else
                    'Add Drawing #5 to our new submenu Note use of Chr function to _
                    insert a quotation at the proper place
                    Set newItem = newMenu.AddMenuItem(intCounter - 4, "GoTo: " _
                        & strCurrDoc, Chr(vbKeyEscape) & Chr(vbKeyEscape) & "vbastmt " _
                        & "Application.documents(" & Chr(34) & strCurrDoc & _
                        Chr(34) & ").Activate" & Chr(10))
                End If
                On Error GoTo 0
                On Error Resume Next
            Case Else
                'Drawing Count More than 5 so just add them to the RtClick Menu
                Set newItem = newMenu.AddMenuItem(intCounter - 5, "GoTo: " _
                    & strCurrDoc, Chr(vbKeyEscape) & Chr(vbKeyEscape) & "vbastmt " _
                    & "Application.documents(" & Chr(34) & strCurrDoc & _
                    Chr(34) & ").Activate" & Chr(10))
                If Err Then
                    'must already exist in popup, skipping now
                    Err.Clear
                End If
            End Select
        End If
    Next intCounter
End If
```

VBA Foundations, Part 11

A Tutorial in VBA for Beginners—The Eleventh of a Twelve Part Series

Richard L. Binning / rbinning@attbi.com

```
        On Error GoTo 0
        On Error Resume Next
    End Select
Else
    Set newItem = ShortcutMenu.AddItem(intCounter + 2, "GoTo: " _
        & strCurrDoc, Chr(vbKeyEscape) & Chr(vbKeyEscape) & "vbastmt " _
        & "Application.documents(" & Chr(34) & strCurrDoc & Chr(34) _
        & ").Activate" & Chr(10))

    If Err Then
        'must already exist in popup, skipping now
        Err.Clear
    End If
    On Error GoTo 0
    On Error Resume Next
End If

Next intCounter 'Keep looping til done
intCounter = intCounter + 2 'add 2 to intcounter to maintain count above our _
initial menu item showing the PCS macro and finally add a separator _
below our new additions if necessary
If Not ShortcutMenu.Item(intCounter).Type = acMenuSeparator Then
    Set newItem = ShortcutMenu.AddSeparator(intCounter)
End If
intMnItemCntAdded = intCounter
End If

End If
End Sub

Private Sub AcadDocument_EndShortcutMenu _
    (ShortcutMenu As AutoCAD.IAcadPopupMenu)
    'Make sure that more than one document is open
    If Application.Documents.Count > 1 Then
        On Error Resume Next
        'Add a dictionary object and find out where the standard _
        top shortcut menu item is located by searching for the item _
        with the value of "&Repeat %s"
        Dim oDictExists As Scripting.Dictionary
```

VBA Foundations, Part 11

A Tutorial in VBA for Beginners—The Eleventh of a Twelve Part Series

Richard L. Binning / rbinning@attbi.com

```
Set oDictExists = New Scripting.Dictionary
Dim intCntNew As Integer
For intCntNew = 0 To ShortcutMenu.Count - 1
    If Not ShortcutMenu.Item(intCntNew).Type = acMenuSeparator Then
        If Not oDictExists.Exists(ShortcutMenu.Item(intCntNew).Caption) Then
            oDictExists.Add ShortcutMenu.Item(intCntNew).Caption, _
                ShortcutMenu.Item(intCntNew).Index
        End If
    Else
        oDictExists.Add "Seperator" & intCntNew, ShortcutMenu.Item(intCntNew).Index
    End If
Next intCntNew

'Declare an integer to get value of count
Dim ix As Integer
Dim intLastValid As Integer
intLastValid = 0

'set value of Default Top Menu item
intLastValid = oDictExists.Item("&Repeat %s") 'ShortcutMenu.Item("&Repeat %s")
Dim msg As String
'Stop 'DevNote

For ix = intLastValid To 0 Step -1
    'Delete the Menu Items added by the StartShortcutMenu Event _
    Probably don't need to do all this checking but if some other _
    developer is adding items here on the fly, then be courteous _
    and try not to step on any toes
    If InStr(1, ShortcutMenu.Item(ix).Caption, "dwg", vbTextCompare) > 0 Then
        ShortcutMenu.Item(ix).Delete
    ElseIf ShortcutMenu.Item(ix).Type = acMenuSeparator Then
        ShortcutMenu.Item(ix).Delete
    ElseIf ShortcutMenu.Item(ix).Type = acMenuSubMenu Then
        If InStr(1, ShortcutMenu.Item(ix).Label, "More", vbTextCompare) > 0 Then
            ShortcutMenu.Item(ix).Delete
        End If
    End If
Next ix

If InStr(1, ShortcutMenu.Item(0).Caption, "dwg", vbTextCompare) > 0 Then
```

VBA Foundations, Part 11

A Tutorial in VBA for Beginners—The Eleventh of a Twelve Part Series

Richard L. Binning / rbinning@attbi.com

```
        ShortcutMenu.Item(0).Delete
    End If
    'Check and see if a separator is in the first item location which _
    is a No No in AutoCAD...trying to be a good citizen here
    If ShortcutMenu.Item(0).Type = acMenuSeparator Then
        ShortcutMenu.Item(0).Delete
    End If
    intMnItemCntAdded = 0
End If
End Sub
```

```
Public Function StripPathFromName(FileName As String) As String
'Note this function uses the built in function "Split" which became
'available with Acad2000i and above.  If you are running Acad2000, which
'has VBA 5, then request the VBA 5 version of Split by email or do a google
'search for the function.
Dim varArFile 'Declare Variant to hold array values
Dim intCnt As Long 'Declare counter to determine count of folders/drives
If Not FileName = "" Then
    varArFile = Split(FileName, "\", -1, vbTextCompare)
    intCnt = UBound(varArFile) - LBound(varArFile)
    StripPathFromName = varArFile(intCnt)
End If
End Function
```

```
Public Sub Close_N_PurgeAllbutCurrent()
On Error Resume Next
Dim objDrawing As AcadDocument
Dim objDwgs As AcadDocuments
Dim strName As String
Set objDwgs = Documents
strName = ThisDrawing.Name
If Documents.Count > 1 Then
    'For each drawing that is currently open in AutoCAD
    For Each objDrawing In objDwgs
        'I would like to automatically switch to each drawing
```

VBA Foundations, Part 11

A Tutorial in VBA for Beginners—The Eleventh of a Twelve Part Series

Richard L. Binning / rbinning@attbi.com

```
        objDrawing.Activate
    'purge the drawing
        objDrawing.PurgeAll
    'save the drawing
        objDrawing.Save
    'Check the drawing name and close
    'every drawing but the current one
        If Not objDrawing.Name = strName Then
            objDrawing.Close
        End If
    'continue doing these steps until
Next objDrawing
Else
    MsgBox "Not designed for Single Documents!", , "Richard Binning Custom Routines!"
End If

'no drawings are open
'then close AutoCAD still not implemented yet

End Sub
```

All right, so now we have a better understanding of what an Event is and what types of Events are available, and we've learned how can we make use of them. For this we turned to our old friend the "Sub Routine" or procedure. A subroutine that was written specifically to respond to an event is called an "Event Handler". The example code shown above for the three types of events was Event Handler type of code. These "Event Handlers" are executed automatically every time their associated event or "trigger" is encountered. This works precisely as one would think. The only other thing we need to know about Event Handlers is that some of the Events, when triggered, will also pass in parameters to the Event handler. Parameters such as the "Command" name, passed by value as a string, are passed into both the "AcadDocument_BeginCommand" and "AcadDocument_EndCommand" event handlers. These parameters are passed in to the associated handlers as a means to provide additional information about the trigger or event that occurred.

In order to use Events in a controlled and safe manner, we must follow some recommendations and rules. Because Event handlers are notifications in real time of events or triggers that occur in the application or document, they are often triggered in the middle of another command or activity so care must be exercised in how we respond to them. The following rules are provided for your use:

- 1. Do not rely on the sequence of events.** *Associated Begin and End triggers will follow each other in the correct order, however, performing a "Save" command may or may not call the "BeginCommand" before the "BeginSave" event. Likewise, the "EndCommand" and "EndSave" events are not guaranteed to follow each other in any consistent manner. You can rely on the fact that a "BeginSave" will always*

VBA Foundations, Part 11

A Tutorial in VBA for Beginners—The Eleventh of a Twelve Part Series

Richard L. Binning / rbinning@attbi.com

fire before an “EndSave” event and a “BeginCommand” will always fire before an “EndCommand” event, just don’t mix them up.

- 2. Do not rely on the sequence of operations.** *Entities and modifications to entities are not always written to AutoCAD’s database in the same order as they were acted on in the Application.*
- 3. Do not attempt any interactive functions from an event handler.** *Simply put, do not try to get information from the user, the application, or use SendCommand functions from inside an event handler.*
- 4. Do not launch a dialog box from within an event handler.** *This rule is similar to #4 above, since a dialog box is typically displayed to gather information. You may display message boxes as notification, though, as most of the online help examples show.*
- 5. Do not write data to the object that issued the event.** *This is a BIG no no! It is already open for write since it is being modified. You can, however, read information from the object in question.*
- 6. Do not trigger the same event.** *We don’t want a sub that calls itself and calls itself and call itself, etc. This can cause an infinite loop that will eventually crash once your memory is exhausted.*
- 7. Remember that no events will be fired while AutoCAD® is displaying a modal dialog.** *Nuff said?*

In summary, AutoCAD® provides some built in notifications called Events, which you, the programmer, can respond to. These events are associated with three categories: Application Level Events, Document Level Events, and Object Level Events. All of these events have associated “Event handlers”, which will be automatically executed when their associated event is triggered by the user or the application. Additionally, we have listed some basic rules to help guide you in using these events successfully.

As always, use the on-line help to further explore these concepts as necessary. If you are really stumped, please send in your questions to the VBA guild or the email address at the top of this article. See you on the guilds or in the next issue of AUGIWorld™ magazine coming soon to a mailbox near you.