

VBA Foundations, Part 12

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

Richard L. Binning / rbinning@comcast.net

“As quickly as you can...Snatch the Pebble from my hand,” he had said as he extended his hand toward you. You reached for the pebble but you opened it only to find that it was indeed still empty. Looking down at your hand you hear, “When you can take the pebble from my hand, it will be time for you to leave.” Upon completion of this last lesson you will have all the tools necessary to snatch that pebble and much more. But before we proceed with this lesson, lets review all that we have accomplished thus far. Remember the quote from Part One?

“My vision for the future is that PC systems will evolve to the point where the user is not even aware that different applications are being invoked to produce a document. One essential element in this vision is a common macro language. A common macro language will have several advantages for users. First, it will be easy to use. Second, it will be the same in a variety of applications. Finally, by using agents—a graphical interface "operative" that can cross applications boundaries—users can work in one application and call parts of other applications into play as needed, or they can start from outside any application and tie them together in various ways...”

—Bill Gates, Chief Software Architect, Microsoft, Inc., Guest Editorial, BasicPro Magazine, 1991

Ponder on that quote while I reassemble the pieces of our VBA puzzle. I began this series with the above quote as a means to demonstrate both the philosophy and enthusiasm behind the visual basic phenomenon. This introductory article paved the way for our further learning by reviewing the history of Visual Basic and introducing and explaining some common terms. The second article continued our exploration of this graphically rich and easy to learn software development environment by introducing the key components of the editor. The third article jumped back into the editor and continued our exploration while further explaining some of the key concepts of the interface. The fourth and fifth articles actually contained some example code utilizing the command line interface and intrinsic objects. It was in the fourth article that our first macro was written. The fifth article explored common programming constructs such as loops, variables, functions, etc. By the midpoint of this series we were deeply exploring the built in functions provided in the VBA language itself and preparing for the transition to AugiWorld magazine. Part seven began the exploration of the “Visual” portion of our studies by looking at Userforms, objects, controls, and demonstrated the usage of some of the implied objects like “Me” which make programming in VBA so much easier. Parts eight and nine were devoted to error trapping and debugging and demonstrating how to best make use of the editor for these purposes. In part 10, we returned to the core concepts of the VBA language, explored the Object Model and the tools used to expose it, and broached the concept of events. The eleventh article explored the very heart and soul of VBA, events, which hold the true power behind VBA.

Of course this series is merely an introduction to VBA and as such cannot be considered a complete and thorough treatment of the subject, but to be thorough in this introduction I must honor my promise and finish by returning to that original Bill Gates quote. The stated goal of a common macro language **has** been realized in the VBA language. This commonality promotes code that can work both inside of and outside of AutoCAD as well as many other VBA enabled applications. Note: there are now hundreds of applications that can be programmed in this way. The key to controlling this type of automation application is a firm understanding of the application’s object model. Be aware that each application’s object model is different, and is programmable to varying degrees. Where some object models are well developed and allow access to virtually every interface and function, others only allow minimal interaction. Remember too that once you become familiar with an object model or two, the rest will fall into place quite easily and rapidly with but a little exploration.

VBA Foundations, Part 12

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

Richard L. Binning / rbinning@comcast.net

Can you picture that pebble in your hand yet? Lets take a look at the references from a few different VBA enabled applications to get familiar with the default object models and type libraries. The default object model or type library for each application is automatically included in any new project at creation. Let's verify this by looking at the default libraries that get referenced by default when we create a new VBA project inside of AutoCAD. We can do that by creating a new project, opening the VBA editor, and left clicking on "Tools --> References" (See Figure 12-1) or by simply opening the Object Browser and reviewing the available libraries. Do you see the available references? Now try the same activity in another application's VBA editor such as Microsoft Excel or Microsoft Word. Notice the similarities? Each editor has a type/object library automatically referenced for use with that application's object model. (Note: as far as we are concerned, a Type Library is a term that is interchangeable with object library.)

"How then do we access other applications", you ask?
"Simple", I respond. "We add a reference to the chosen application's object model or type library". For example, to create a reference to Microsoft Excel select the check box next to "Microsoft Excel #.0 Object Library." Note: replace the pound symbol in the previous example with the number corresponding to your version of Excel. (8 = Excel 97, 9 = Excel 2000, 10 = Excel 2002, etc.) Now to control that application, we create an object to hold an instance of the application and link to that object in the following fashion.

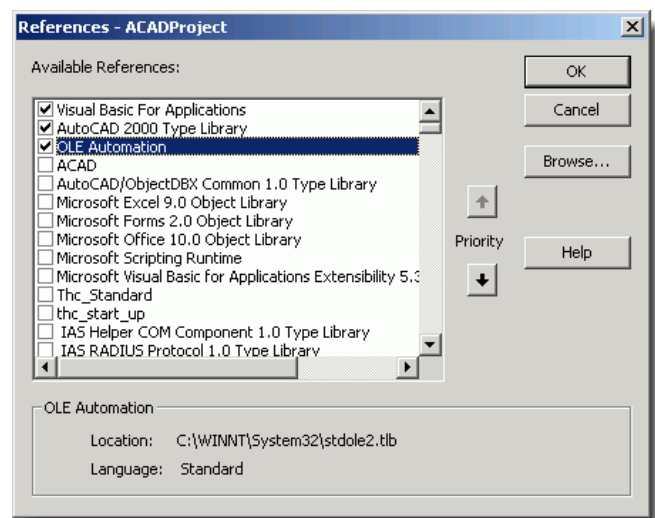
```
Dim oApp as Excel.Application
```

```
Set oApp = New Excel.Application
```

The "Dim" keyword creates or allocates a spot in memory big enough to hold the application object. Note that you haven't really done anything yet, you've simply told the VBA editor that you intend to utilize some memory. The new instance of Excel is actually created when the "New" keyword is acted upon in the "Set" statement.

Note: *this is the preferred method of creating an instance of an automation server because it gives you the greatest amount of control over when the instance is actually created.*

Now you can begin using your instance of Excel from within AutoCAD just like you were working in Excel itself. The method for creating an instance of other automation servers is similar. So what can we do with our instance of Excel? Lets try something. Start a new VBA project in AutoCAD; I won't direct you since you already know how to do this. Lets make sure that we add a reference to our version of Excel. Now create a public sub routine and lets create a link to Excel. First thing we need is to create the skeleton of our error handler. Then we can fill in the gaps. What else do we need? How about some variables? Lets create some string variables to handle messages and names and throw in a couple of object variables to hold Excel specific entities such as the application, a collection of recent files, and an iterator object to cycle through the recent files. With these minimal variables and objects we can launch Excel, grab the application, get a hold of the recent files collection, cycle through the collection, shut down Excel, and return a message to us within AutoCAD. Take a look at figure 12-2. Does your code look like this? (See Figure 12-2).



(Figure 12-1)

VBA Foundations, Part 12

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

Richard L. Binning / rbinning@comcast.net

```
Public Sub TestConn2Excel()  
    'Set up a minimal error catcher  
    On Error GoTo Err_Catcher  
    'Declare some variables  
    Dim strMessage As String  
    Dim strApplicationName As String  
    Dim i As Integer  
    Dim oApp As Excel.Application 'Set aside memory for the excel application  
    Dim oRecentFile As Excel.RecentFile 'Lets readback a recent file  
    Dim oRecentFiles As Excel.RecentFiles 'Lets read them all  
    Set oApp = New Excel.Application 'Create the instance  
    'Note Excel should now be visible on your task bar  
    oApp.Visible = True 'We will make it visible  
    'lets minimize it so it sits comfortably in the task bar  
    oApp.WindowState = xlMinimized  
    MsgBox "Look at your Task Bar...you should see Excel running!", _  
        vbInformation, "Excel Connect!"  
    Set oRecentFiles = oApp.RecentFiles 'Grab the collection of recent files  
    'For each file in the collection grab some data  
    For Each oRecentFile In oRecentFiles  
        i = i + 1  
        strMessage = strMessage & vbCrLf & i & ".) " & _  
            & oRecentFile.Name & " Launched by: " & oApp.UserName  
    Next oRecentFile  
    'Formulate reply message  
    strApplicationName = oApp.Caption  
    strMessage = "Successfully launched " & strApplicationName & vbCrLf & _  
        "and detected the following recent files:" & strMessage  
    'Excel objects must be released in reverse order from their creation  
    Set oRecentFiles = Nothing  
    Set oApp = Nothing  
    'Note excel should now be missing from your task bar  
    'display the message  
    MsgBox strMessage, vbInformation, "Message from " & strApplicationName  
Err_Exit:  
    Exit Sub  
Err_Catcher:  
    MsgBox Err.Number & vbCrLf & Err.Description, _  
        vbCritical, "Error!"  
    Resume Err_Exit  
End Sub
```

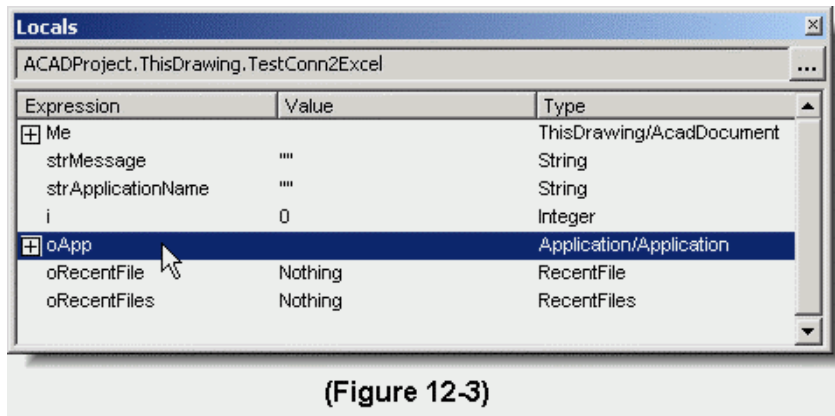
(Figure 12-2)

Now it is your turn to explore, instead of just clicking the run button, try stepping through this code a line at a time. Once you have made the connection, open the “Locals” window and explore the oApp object. You will see a plethora of objects and properties you can access, modify and use to your hearts content. Take a look at Figure 12-3 for information on where to look for objects and properties available to you, the Excel master, while working with this spread sheet application in a programmatic manner. (See Figure 12-3)

VBA Foundations, Part 12

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

Richard L. Binning / rbinning@comcast.net



(Figure 12-3)

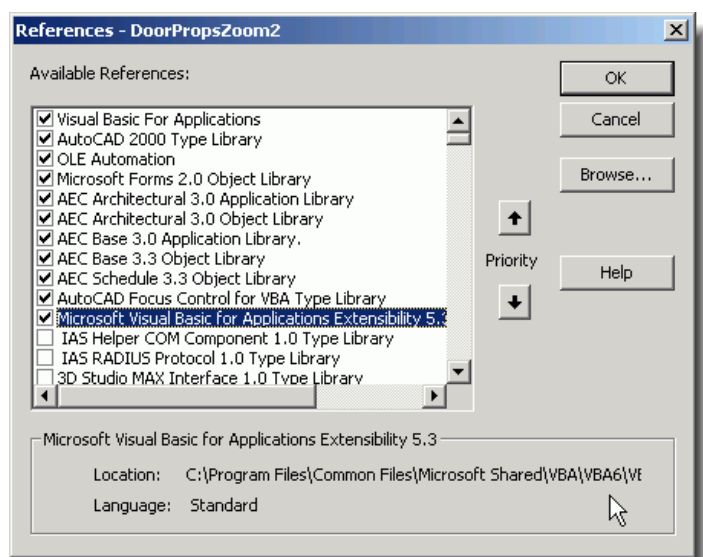


Note: one of the idiosyncrasies of working with Excel is you must always remember to remove objects in the reverse order of their creation. Otherwise you may leave a session of Excel running invisibly in the background.

Once you get the hang of working with Excel, you will find that the other VBA enabled applications that are part of Microsoft Office will work in a very similar manner.

Lets take a look at some routines written in one of AutoCAD’s “vertical” applications. In this example we will access AutoCAD Architectural Desktop and create a routine that will search for a particular door number. If found, then the routine will perform a “Zoom” Center command to display the door in the center of your work screen.

To get started with this example: We must first add some references. Start a new project and call it Zoom2Door. In this project we will have a module and a userform. Within your project open the Tools menu and choose references. Make sure that your list of references matches the reference dialog box shown in Figure 12-4. In this project we are going to make use of the VBA Extensibility library as well as some of the Architectural Desktop libraries. In addition, we will utilize an undocumented custom control provided for your use by the programmers at Autodesk. This control may or may not be already registered on your system. When you go to add it to the toolbox, if it is not listed you should browse to the following location and select it. C:\Program Files\Common Files\Autodesk Shared\AcFocusCtrl.dll. Add this control by dragging it onto your form and giving it the



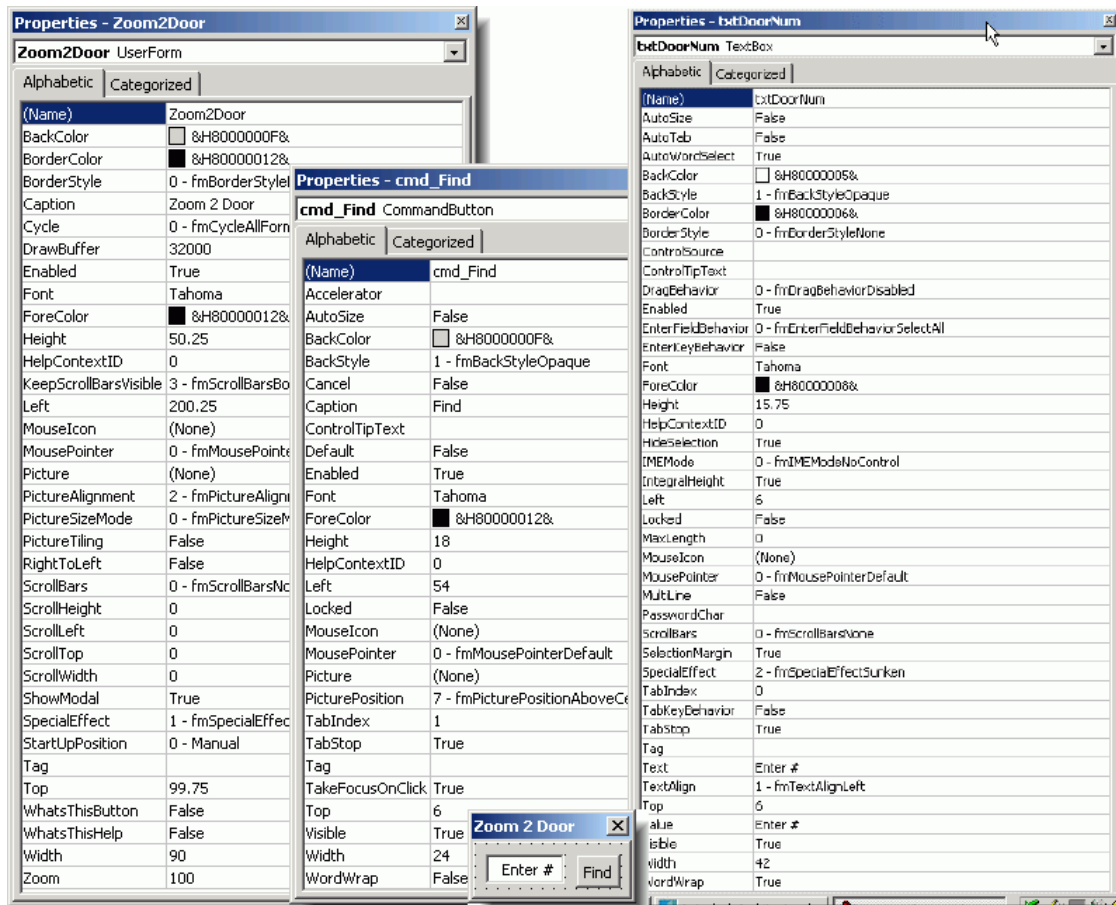
(Figure 12-4)

VBA Foundations, Part 12

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

Richard L. Binning / rbinning@comcast.net

following name: “AcFocusCtrl1”. If you have to browse for this control, then chances are you will have to register the dll for your use prior to running this example. Do a google search on the use of regsvr32.exe. If you get stumped, drop me a line and I’ll guide you through the registration process.



(Figure 12-5)

Once you have the references added to your project, we can add the following routines. Select your UserForm and switch to the code window. Ensure that we are working in Option Explicit mode. We will also set “Option Compare Text” so that we can work without regard to upper versus lower case text in our searches. Since we are using the AutoCAD focus Control for VBA, we must also program some events to control it. This control allows AutoCAD users to interact with the AutoCAD menu, drawing window, and objects while the form is still being displayed on top of AutoCAD’s window. Because of this we will want to be able to set focus, keep focus, and relinquish focus. To that end you must also add a userform to your project and name it “Zoom2Door”, but don’t include the quotation marks. Add a textbox to this form and a command button. You should name the textbox, “txtDoorNum”, again without the quotes. The Command button’s name is “cmd_Find”. See Figure 12-5 for the layout I used. Once we have the form in place we will write code to handle our form during the following events: “txtDoorNum_MouseDown”, “cmd_Find_Click”, and “UserForm_MouseMove”. (See Figure 12-6) We will wish to be able to search for our door objects both within the current drawing and within any external references that may be attached to the current drawing. For this we will add a subroutine to our form to do the current

VBA Foundations, Part 12

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

Richard L. Binning / rbinning@comcast.net

drawing search. If that search fails, then we will call a function to search within any xrefs attached to the current drawing. Because this is the last place we search, if no door with the desired number is found, we will respond with a message to the user telling him/her that the door doesn't exist. The last function we will need will demonstrate a method of unloading the current VBA macro using a command line message. Please review and enter the following code (See Figures 12-6 through 12-10B) making sure to check for syntax errors or misspellings. Note: I have heavily commented the following code so I will let you read the comments for an explanation rather than duplicating the effort here.

The following code should be added to the UserForm named Zoom2Door:

```
Private Sub cmd_Find_Click()  
    'The AcFocusCtrl allows us to maintain visibility of the form while  
    'we interact with the drawing entities...this control fixes AutoCAD's broken  
    'implementation of the Modal dialog box  
    'Since we desire to check for a door we can now remove focus from the form  
    Zoom2Door.AcFocusCtrl1.KeepFocus = False  
    'Call the main sub to find the door number  
    FindDoor  
End Sub  
  
Private Sub txtDoorNum_MouseDown(ByVal Button As Integer, ByVal Shift As Integer, _  
    ByVal X As Single, ByVal Y As Single)  
    'Lets maintain focus and clear the textbox for use  
    'This is the specific call to the control found on the Zoom2Door UserForm object  
    Zoom2Door.AcFocusCtrl1.KeepFocus = True  
    txtDoorNum.Text = ""  
End Sub  
  
Private Sub UserForm_MouseMove(ByVal Button As Integer, ByVal Shift As Integer, _  
    ByVal X As Single, ByVal Y As Single)  
    'Gather focus to the form anytime a mouse travels within it's boundary  
    'This is the generic call to the control found on the current object  
    Me.AcFocusCtrl1.SetFocus  
End Sub  
  
Private Sub UserForm_QueryClose(Cancel As Integer, CloseMode As Integer)  
    'Prevent user from closing with the Close box in the title bar.  
    If CloseMode = 0 Then Cancel = 1  
    Unload Me  
    Cancel = 0  
    'Set a flag for other routines  
    ThisDrawing.SetVariable "USER15", 9  
    Dim tmpName As String  
    tmpName = "_vbaunload" & ProjFullPath  
    'Self unloading routines using send command must call the command  
    'absolutely last  
    ThisDrawing.SendCommand tmpName  
End Sub
```

(Figure 12-6)

VBA Foundations, Part 12

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

Richard L. Binning / rbinning@comcast.net

```
Sub FindDoor()  
  'Declare Objects for search  
  Dim acadobj As AcadObject  
  Dim Sched As New AecScheduleApplication  
  Dim PropSets As AecSchedulePropertySets  
  Dim PropSet As AecSchedulePropertySet  
  Dim Prop As AecScheduleProperty  
  Dim doorCnt As Integer  
  doorCnt = 0  
  'ensure that our boolean is preset each time  
  blnFlag = False  
  Dim doornum As String  
  'Need some generic iterators  
  Dim i As Integer  
  Dim ii As Integer  
  'Set up some initial error checking  
  On Error GoTo Goodbye  
  'Doors are bound to be found in ModelSpace  
  For Each acadobj In ThisDrawing.ModelSpace  
    'Check for type of Object...hint this is the other way to check  
    If acadobj.ObjectName Like "AecDbDoor" Then  
      'This isn't being used yet  
      doorCnt = doorCnt + 1  
      'Gather data from Object data is the attached property sets  
      Set PropSets = Sched.PropertySets(acadobj)  
      'Cycle through the property sets attached  
      'could we use a for each here...you tell me  
      For i = 0 To PropSets.Count - 1  
        'Set each object for checking  
        Set PropSet = PropSets.Item(i)  
        'Check for ADT21 or ADT33 AecDoor Object type of Property set  
        If PropSet.Name = "Door" Or PropSet.Name = "DoorObjects" Then  
          For ii = 0 To PropSet.Properties.Count - 1  
            'Set each property and check its name value  
            Set Prop = PropSet.Properties.Item(ii)  
            If Prop.Name = "#" Or Prop.Name = "Number" Or Prop.Name = "01-NUMBER" Then  
              'Now we are ready to compare values  
              'Get value of this door  
              doornum = Prop.Value  
              'Check if this is the door  
              If doornum Like Me.txtDoorNum.Text Then  
                blnFlag = True  
                Dim min As Variant ' we will store a location for the door on the tag  
                Dim max As Variant ' use the bounding box to find the extents of the door  
                acadobj.GetBoundingBox min, max ' get the bounding box for the door  
                Dim tempCntrPt(0 To 2) As Double ' center of the bounding box  
                tempCntrPt(0) = (min(0) + max(0)) / 2#  
                tempCntrPt(1) = (min(1) + max(1)) / 2#  
                tempCntrPt(2) = 0  
                Dim zoomCntrPt As String  
                zoomCntrPt = tempCntrPt(0) & "," & tempCntrPt(1) & "," & tempCntrPt(2)  
                ThisDrawing.SendCommand CStr("mspace zoom c " & zoomCntrPt & " 250 ")  
                GoTo Goodbye  
              End If  
            End If  
          Next ii  
        End If  
      Next i  
    End If  
  Next acadobj  
  'MsgBox "Count of Doors " & doorCnt  
  Goodbye:  
  If blnFlag = False Then  
    'Check Xref  
    If Not CheckXref(txtDoorNum.Text) Then  
      MsgBox "Door #" & txtDoorNum.Text & " does not exist!", vbInformation  
    End If  
  End If  
  'Do Cleanups  
  Set Prop = Nothing  
  Set PropSet = Nothing  
  Set PropSets = Nothing  
  Set Sched = Nothing  
End Sub
```

(Figure 12-7)

VBA Foundations, Part 12

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

Richard L. Binning / rbinning@comcast.net

The following code should be added to the module window in your project:

```
Option Explicit
Public blnFlag As Boolean
Sub Run_dr()
    'This is the public sub routine that is required
    'so that we can call the routine from outside the
    'VBA editor from either the vbamanager or via lisp
    'Note I could have left this as UserForm1 and
    'loaded it using Me.Show, but this gives us more
    'options in an environment that may be running
    'multiple vba routines at the same time.

    'Note: Modeless means that we can click and interact with
    'the autocad window without hiding the form.
    Zoom2Door.Show vbModeless
End Sub
```

(Figure 12-8)

```
Public Function ProjFullPath() As String

    'We can grab the "mapped" drive path of the current VBA macro
    'by using the VBA extension and get the path from the caption
    'of the current macro
    'Note: The implementation of the unload command does not allow
    'usage of UNC path names..so we use this kludge to return the
    'Mapped drive name and path
    Dim oVBE As VBIDE.VBE
    Dim sFullPath As String
    Set oVBE = ThisDrawing.Application.VBE
    sFullPath = VBA.Mid(oVBE.MainWindow.Caption, 25)
    sFullPath = VBA.Left(sFullPath, InStr(1, sFullPath, "[") - 1)
    ProjFullPath = sFullPath

End Function
```

(Figure 12-9)



Note: The VBA extensibility library has many uses. Take some time to explore this library and see what it offers.

VBA Foundations, Part 12

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

Richard L. Binning / rbinning@comcast.net

The following code has been broken up so that readability could be enhanced. Please ensure that you have copied all the code.

```
Function CheckXref(doorNum1 As String) As Boolean
    'Declare variables for our Schedule and Property set objects
    'We will be checking Xrefs in this Function
    Dim Sched As New AecScheduleApplication
    Dim PropSets As AecSchedulePropertySets
    Dim PropSet As AecSchedulePropertySet
    Dim Prop As AecScheduleProperty
    'We'll get the number from the property set
    Dim doornum As String
    'Our iterator the trusty Acad Object
    Dim acadobj As AcadObject
    'We need some generic interator counters for those objects
    'that do not provide their own collections
    Dim i As Integer
    Dim ii As Integer
    Dim xRefDrawing As AcadBlock
    'Because an XRef is really a type of block
    Dim xRefDrawingBlock As AcadBlock
    'Look at each block in the current drawing
    For Each xRefDrawing In ThisDrawing.Blocks ' look at the xrefs in the drawing
        'If it is an external Reference then we will check for our number
        'The line following is wrapped in parenthesis because it is really
        'a type of inline function that will return either "True or False"
        'The result could be read as If True Then...
        If (xRefDrawing.IsXRef) Then
            ' look at the blocks in xrefs in the drawing
            For Each xRefDrawingBlock In xRefDrawing.XRefDatabase.Blocks
                'Now lets check each object in this block
                For Each acadobj In xRefDrawingBlock
                    'Check the object type using the TypeOf method which
                    'returns the Class type
                    If TypeOf acadobj Is AecDoor Then
                        'Pass a validated object to gather
                        'the attached property sets for the
                        'object, in this case an ADT door
                        Set PropSets = Sched.PropertySets(acadobj)
                        For i = 0 To PropSets.Count - 1
                            'Base Zero array so adjust by subtracting 1
                            Set PropSet = PropSets.Item(i)
                            'Check for ADT2i or ADT33 Door Property Set
                            If PropSet.Name = "Door" Or PropSet.Name = "DoorObjects" Then
```

(Figure 12-10A)

VBA Foundations, Part 12

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

Richard L. Binning / rbinning@comcast.net

```
For ii = 0 To PropSet.Properties.Count - 1
Set Prop = PropSet.Properties.Item(ii)
'Check for ADT2i or ADT33 or Custom Property
If Prop.Name = "#" Or Prop.Name = "Number" Or Prop.Name = "01-Number" Then
'Assign the door number for a check
doornum = Prop.Value
'Check using Like with Option Compare Text
If doornum Like doorNum1 Then
'Set flag for check from previous function
blnFlag = True
'We could have declared these ahead of time, but _
we only need them once and we don't need them unless _
we find the door, so here are the declarations
Dim min As Variant ' we will store a location for the door on the tag
Dim max As Variant ' use the bounding box to find the extents of the door
acadobj.GetBoundingBox min, max ' get the bounding box for the door
Dim tempCtrPt(0 To 2) As Double ' center of the bounding box
tempCtrPt(0) = (min(0) + max(0)) / 2#
tempCtrPt(1) = (min(1) + max(1)) / 2#
tempCtrPt(2) = 0
Dim zoomCtrPt As String
zoomCtrPt = tempCtrPt(0) & "," & tempCtrPt(1) & "," & tempCtrPt(2)
'perform a zoom center looking at our door
ThisDrawing.SendCommand CStr("mspace zoom c " & zoomCtrPt & " 250 ")
'Exit Function
GoTo CheckExit
End If
End If
Next ii
End If
Next i
End If
Next
End If
Next
'Do cleanups
Set Sched = Nothing
Set PropSets = Nothing
Set PropSet = Nothing
Set Prop = Nothing

CheckExit:
If blnFlag Then
GoTo CleanExit
End If
'Number not found so return value and exit
CheckXref = False
Exit Function

CleanExit:
CheckXref = True
End Function
```

(Figure 12-10B)

This routine will be available in its entirety for your use by download at the Augi Exchange site by the time you read this. Please do a search for Zoom2Door. Also take the time to explore other routines, symbols, and papers available for your use at the Augi Exchange site.

As always, use the on-line help and the additional resources presented in this article to further explore these concepts as necessary.

VBA Foundations, Part 12

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

Richard L. Binning / rbinning@comcast.net

Look at your hand now...slowly open it and turn it so that your palm is facing the sky...isn't that the pebble you were reaching for. It must be time for you to leave, Grasshopper. Thank you for following this series. I have enjoyed writing it.

See you on the guilds.