

VBA Foundations, Part 7

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

Richard L. Binning / rbinning@attbi.com

Welcome to this months edition of VBA Foundations in its new home as part of AUGIWorld™. This document is the "full version" of the article that appears in the September/October issue of Augiworld magazine, in which we are limited to two pages per article. In this issue we are going to explore VBA's forms and see how easy it is to create a graphically rich and professional looking program.

How often in your work have you may wished to pop up a message or a fancy dialog box while your program was running? Have you ever tried this in other programming languages? Autolisp comes to mind, and just as quickly exits stage left. One of the primary reasons most people wish to learn VBA is to be able to create a graphically rich program without writing pages and pages of code to display the forms. Lets take this opportunity to re-visit the quote from Bill Gates that appeared in the first issue of this series. "...*Imagine designing the visual components of an application graphically—merely by placing controls on a form. All programs would be designed, created and run within the Windows environment. Taking the ease of use that is common to both BASIC and Windows, the result would be a visual, productive, and interoperable tool for creating applications in the environment that is currently the most popular for personal computers.*" (Bill Gates, Chief Software Architect, Microsoft, Inc.)¹ Yes, let that sink in because that is exactly how we are going to create our visual forms and dialogs for our programs by utilizing VBA.

According to the help file that is installed with the VBA editor in AutoCAD 2002, the definition of a form is as follows: **Form** - *A window or dialog box. Forms are containers for controls. If you have been following this series, then you know that the first bit of code we explored utilized one of VBA's built in forms called the Message box. In that issue I showed how easy it was to invoke a form directly from the command line in AutoCAD without ever-opening the VBA editor. That is the reason why the thought of Autolisp makes my blood run cold. Actually, it is the thought of writing all that DCL code to display a form for use with AutoLisp that does it. But I digress, command line generated forms are fine for displaying standard forms, but what if you want to create your own user interface or just make your programs more professional and "Windows" like? Then pay attention here because we are going to explore the use of Forms in VBA. If you have the VBA editor opened with a project loaded then you can add a form by simply mousing over to the project window, right click on the name of your project, and select "Insert -> UserForm". Another way to insert a Form is to travel up to your menu bar and select "Insert", then "UserForm". "Of course there is!" I respond after being asked if a form can be inserted without touching the mouse. Try this for yourself, hold your "Alt" key down and select the following keys in rapid succession "I" then "U". I *knew* you old timers would appreciate that. Another method to insert a form requires having a form already created, perhaps with some controls in place. You may have noticed the "Import" option available when you right click on your project in the project window. You can choose this option if you already have a form saved. This same option is also available from the "File" menu. If you don't have the editor open, now is as a good a time as any to begin exploring. So lets begin by opening our last project "Purgall.dvb". You can find the code, fully detailed and explained, in a previous issue. I list it here for those latecomers who may have missed the previous issues. (See Code Below)*

```
Public Sub Close_N_PurgeAll()
```

```
On Error Resume Next
```

VBA Foundations, Part 7

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

Richard L. Binning / rbinning@attbi.com

```
Dim objDrawing As AcadDocument
Dim objDwgs As AcadDocuments
Set objDwgs = Application.Documents
```

```
'For each drawing that is currently open in AutoCAD
```

```
For Each objDrawing In objDwgs
```

```
    'I would like to automatically switch to each drawing
```

```
        objDrawing.Activate
```

```
        'purge the drawing
```

```
        objDrawing.PurgeAll
```

```
        'save the drawing
```

```
        objDrawing.Save
```

```
        'close the drawing
```

```
        objDrawing.Close
```

```
    'continue doing these steps until
```

```
Next objDrawing
```

```
'no drawings are open
```

```
'then close AutoCAD
```

```
End Sub
```

Listing 1 (Place this in “ThisDrawing” in Purgeall.Dvb)

Now utilize one of the methods I mentioned earlier to insert a form into this project. Ready? Lets learn some more about forms. User forms have properties, events, and methods that determine their appearance such as position, size, and color; and control aspects of their behavior both by themselves and in response to events. The size and location of a form is defined by manipulating the following properties of forms: Top, Left, Width, Height. And where are these properties located? Please, take this opportunity to explore all the properties available to you as the creator of a form as I proceed to list them for you below. You can locate a form in AutoCAD’s active window by setting the value of Top and Left to the actual screen location. *(Note: when you first insert a form into your project, its default placement is most likely going to be set to “1 – CenterOwner”.* You can check this by looking for the

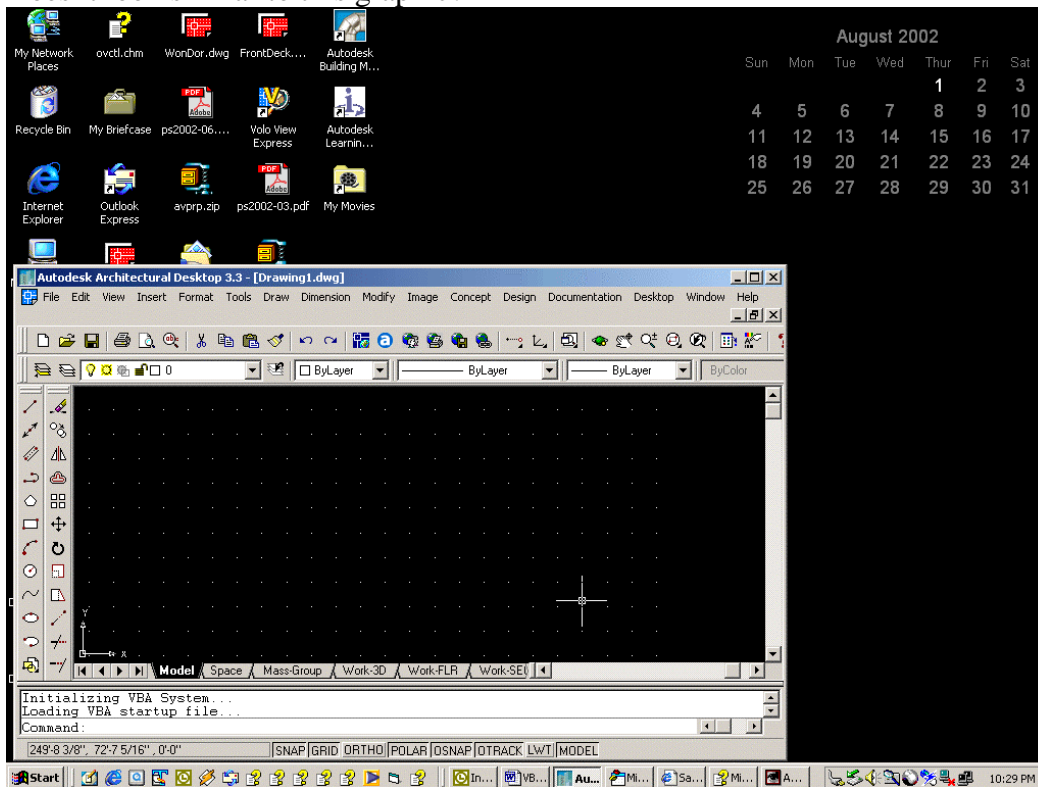
VBA Foundations, Part 7

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

Richard L. Binning / rbinning@attbi.com

“StartupPositionProperty”). These placement properties are defined as the distance between a form and the left or top edge of the form or window that contains it. Note this is typically a number between -32,767 to +32,767 on most Windows systems, although the recommended range is only from 0 to +32,767. The values for Width and Height are simply added to Top and Left to define the bottom and right side of the form. The actual numbers are called points. VBA uses points to measure the size and location of forms and objects. You are probably already familiar with points since all the font sizes you use in other Windows programs. This measurement equates to 1/72 of an inch.

Lets see what effect some of the properties of the form have. Switch to your AutoCAD window, move the mouse to the right corner of the screen just above and to the right of the close button (the ever-present “x” inside the square). When your mouse cursor changes to a double-sided arrowhead use your left mouse button to click, now drag the AutoCAD window until it no longer takes up the whole screen. Does it look similar to this graphic?



Now switch back to the VB editor. In the Project Explorer select your UserForm object. Click the “F5” key or press the run button located on the VB editor tool bar (it is the right facing Triangle). Notice that the form immediately responded by displaying itself in the center of AutoCAD’s window. Now click the “x” button at the top right of the form to close it. Change the “StartupPositionProperty” to “2 – CenterScreen”. Now run it again. Did you see the difference? Try the other options out for yourself.

You, the programmer, can also affect the Borderstyle, BorderColor, BackColor, and Forecolor of the form. Forms can have a standard font associated with them and are typically displayed with a Caption

VBA Foundations, Part 7

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

Richard L. Binning / rbinning@attbi.com

Bar or Window Title bar at the top of the form. This bar also contains a changeable “caption” property in addition to the exit button (an x inside of a square button at the upper right most corner) that is useful for exiting your routine when complete. Other features of the UserForm include 5 different styles of special effects for the form: Flat, Raised, Sunken, Etched, and Bump. Additionally, you can change the mouse cursor as it appears to the user when they have moved the mouse within the borders of your form. To access these properties at design time simply activate your form and look at the properties window for all the properties listed and a few esoteric ones that I’ll leave to your watchful eye. You can also access these properties at runtime (during execution of your form) by declaring their properties in your code. For example to change the special effect of your form to appear sunken with a raised border while it is running you might call the following line of code somewhere in your program:

UserForm1.SpecialEffect = fmSpecialEffectSunken (*Note: you can replace “UserForm1” with the keyword “Me” if the code is part of the form’s code window.*)

UserForms also have methods associated with them. Methods of UserForms refer to the actions that the forms can perform. Actions UserForms can perform include loading and unloading, showing and hiding, and printing. Additionally, UserForms are automatically linked to the help system so that the WhatsThisMode method can be invoked and used to display context sensitive help for the UserForm itself. Note: this is the same functionality available to almost all Microsoft Windows applications available by clicking the question mark button located at the top right of most dialog boxes. (Got a form and it shows no question mark at the top right? Check out your form properties and set the “WhatsThisButton” property to True!)

UserForms Methods are called during execution of your program and can be accomplished within the Form’s Code window by starting your code with the keyword “Me”. To access methods of the UserForm from the ThisDrawing class or any other classes or modules simply start your call by typing in the name of the UserForm as shown in the following example.

UserForm1.Show (This will cause your form to pop to the front of all other forms, windows, etc in the current session of Windows.)

UserForm1.Hide (This will cause your form to disappear from view. Note: the form is still loaded in memory and the code is able to execute it is simply hidden from the users view.)

UserForm1.Load (Make note of the following information from the VBA help file concerning this method: *When an object is loaded, it is placed in memory, but isn't visible. Use the **Show** method to make the object visible. **Until an object is visible, a user can't interact with it. The object can be manipulated programmatically in its Initialize event procedure.***)

UserForm1.Unload (Again, please make note of the following information regarding this method: *When an object is unloaded, it's removed from memory and all memory associated with the object is reclaimed. Until it is placed in memory again using the **Load** statement, a user can't interact with an object, and the object **can't** be manipulated programmatically.*)

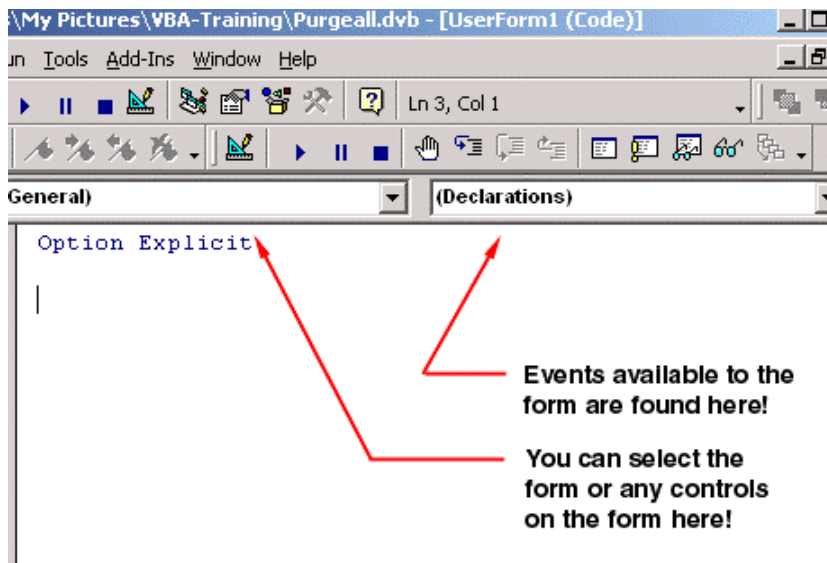
VBA also enables Forms to interact act with your code or respond to events that occur during the execution of your project. Events available to your form can be found by looking at the code window

VBA Foundations, Part 7

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

Richard L. Binning / rbinning@attbi.com

behind your form. At the top right side of the UserForm code window is a pulldown or combobox control containing all the default events available for your UserForm to react to. (See Graphic)



To see the following Events in your own Code Window, switch to the code window as shown above. Now using the combobox on the left to select the “UserForm” Object. Now click the right combobox to display the events available. Events include: Activate, AddControl, BeforeDragOver, BeforeDropOrPaste, Click, DblClick, Deactivate, Error, Initialize, KeyDown, KeyPress, KeyUp, Layout, MouseDown, MouseMove, MouseUp, QueryClose, RemoveControl, Resize, Scroll, Terminate, and finally Zoom. The most frequently used events are probably QueryClose, Initialize, and MouseMove. These events are already defined and simply waiting for you to add some code to them to make them useful. To utilize them simply switch over to the code window for the UserForm, make sure that the “Object Box” is showing the words “UserForm” and then use the combobox on the right side to select your event of choice. As soon as you make your selection the “boiler plate” code is inserted into the code window and you are able to add your own commands within the event function.

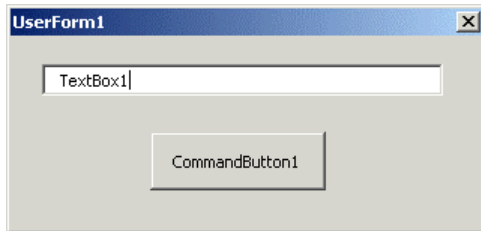
Now lets work with our Form. In our original project, “Purgeall.Dvb”, we have three commands: .Purgeall, .Save, and .Close. But what if we wanted to purge, save and close every drawing but one? How would we accomplish this? In order to interact with this project we will make use of our new form. So lets decide what we want to accomplish. Our code was written by following this bit of logic in the original article. *“For each drawing that is currently open in AutoCAD, I would like to automatically switch to each drawing, purge the drawing, save the drawing, close the drawing, continue doing these steps until no drawings are open then close AutoCAD.”* We began by placing this psuedo code in our code window and building our function or sub routine around it. So if we think about what we want to do, we realize that we need to modify our original code in a few places and setup our form to respond to a selection made by our user. We are going to allow our user to select the drawing to keep open, purge it and save it and then proceed to purge, save, and close all the other drawings. How do we get the name of the drawing we wish to keep open?

VBA Foundations, Part 7

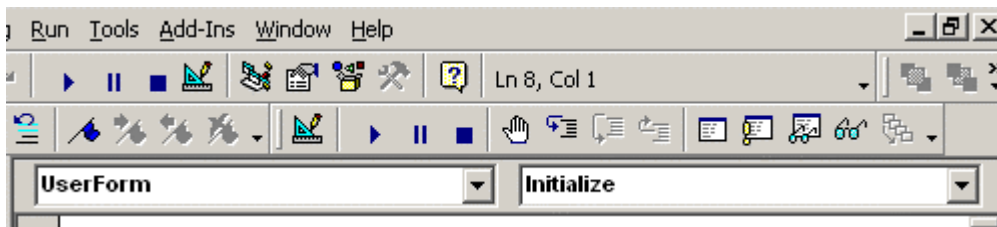
A Tutorial in VBA for Beginners—The Seventh in a Twelve Part Series

Richard L. Binning / rbinning@attbi.com

Option 1. Have the user type in the name. (We could do this, by providing a “textbox” control on our form. What happens if the user mistypes the name?) We could do this by inserting a form into our project. Drag a textbox onto the form from the control toolbox along with a command button. The form looks like this in design mode:



To make the project more standard we will set the Form’s caption and command button caption at runtime by setting them in the form “Initialize” event. I will discuss these events in a later article in more depth so for now just bear with me and place it in the initialize event. To do this we can select our “UserForm” object in the object box and click the right combobox and select “Initialize” from the dropdown list. (See Graphic Below)



This action will automatically insert the Event Code into the code window, so we will click on the first line below “Private Sub UserForm_Initialize()” and begin our manipulations. (See affected code below.)

```
Private Sub UserForm_Initialize()  
    'Using the "Me" keyword to refer to our  
    'User form lets set the caption  
    Me.Caption = "Purge All But TextBox"  
    'Clear the textbox  
    Me.TextBox1.Text = ""  
    'Set the Command Button to say something  
    'besides "CommandButton1"  
    Me.CommandButton1.Caption = "Purge Em"  
End Sub
```

Since we will *learn by modifying* existing code, lets grab our previous code (as shown previously) from the “ThisDrawing” object and use it for this option by placing it into the form

VBA Foundations, Part 7

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

Richard L. Binning / rbinning@attbi.com

code window. We need to copy every line starting with “On Error Resume Next” and stopping just prior to “End Sub” and place them into a new subroutine. Now we place our check on the drawing name (hopefully typed into the text box correctly by our user) in the code below. (As Shown Highlighted below)

```
Private Sub Close_N_PurgeAllButTextBoxSelection()  
    On Error Resume Next  
    Dim objDrawing As AcadDocument  
    Dim objDwgs As AcadDocuments  
    Set objDwgs = Application.Documents  
    'For each drawing that is currently open in AutoCAD  
    For Each objDrawing In objDwgs  
        'I would like to automatically switch to each drawing  
        objDrawing.Activate  
        'purge the drawing  
        objDrawing.PurgeAll  
        'save the drawing  
        objDrawing.Save  
        'Check the drawing name and close  
        'every drawing but the textbox name  
        If Not objDrawing.Name = Me.TextBox1.Text Then  
            objDrawing.Close  
        End If  
        'continue doing these steps until  
    Next objDrawing  
End Sub
```

Whats missing? That’s right, what happens when we click the command button? Nothing, we haven’t added the code yet. Switch to the VBE and double click on the “UserForm” in the project explorer window. Now with the form clearly visible in the editor, double click on the command button. You will be switched to the click event code automatically inserted into your form ready for your additions. I heard that! You’re right! This is too easy! Add the following code between the lines of code just created by your double click and we are ready to try it out.

VBA Foundations, Part 7

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

Richard L. Binning / rbinning@attbi.com

```
Private Sub CommandButton1_Click()  
    Close_N_PurgeAllButTextBoxSelection  
End Sub
```

Option 2. Default to keep the currently active drawing open and close all others. (This is probably the easiest of the options. We could declare a variable to hold the name of the currently active drawing and each time we access one of the open drawings check the name and if it doesn't match our saved name then we close it.) One variable declaration, and an if then condition prior to the “.Close” method and mission accomplished. (See Modified/Highlighted Code below)

```
Public Sub Close_N_PurgeAllbutCurrent()  
    On Error Resume Next  
    Dim objDrawing As AcadDocument  
    Dim objDwgs As AcadDocuments  
    Dim strName As String  
    Set objDwgs = Application.Documents  
    strName = ThisDrawing.Name  
    'For each drawing that is currently open in AutoCAD  
    For Each objDrawing In objDwgs  
        'I would like to automatically switch to each drawing  
        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  
  
    'no drawings are open
```

VBA Foundations, Part 7

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

Richard L. Binning / rbinning@attbi.com

```
'then close AutoCAD
```

```
End Sub
```

This option would probably be sufficient for our purposes, but we want a “True” Windows style “Professional” interface don’t we? Plus we are still forcing the user to switch to the drawing we wish to remain open prior to running the routine.

Option 3. Allow the User to select the Drawing Name to leave open and make our project more “Windows” like and more professional in appearance. I leave this option to you for homework. You should be able to use the code I’ve given you to accomplish this option. Hints: use a combo box instead of a text box. Fill the combo box with the drawing names during the initialize event using the following format. Don’t forget to declare the objects you’ll need and populate any collection.

```
For Each objDrawing In objDwgs  
    ComboBox1.AddItem objDrawing.Name  
Next objDrawing
```

Now you can use the same check as the first option but you must change the control property to match your selection from the combobox like this. Instead of `TextBox1.Text` try substituting `ComboBox1.Value`.

I hope you have found this exploration of VBA forms useful and informative. As always, please spend some time looking through the help whenever you have a question related to the use of this new information. If you are really stumped, feel free to send in your questions and comments to the email address at the top of this article. If you are not already a guild member please join the VBA guild and others. These guilds are in place for you and all are welcome both beginners and experts alike. See you on the guilds or next issue in AUGIWorld™ magazine on your doorstep.

ⁱ Guest Editorial, BasicPro Magazine, 1991