

VBA – Build Your Own Color Picker

by Mark Stefanchuk, Cadgurus.com

Our Early MicroStation VBA Programs

Many early MicroStation VBA programs our studio developed had very simple interfaces, especially in the preference user forms, a place to make command settings such as color, level, and weight settings.

In our first couple applications, our text boxes only allowed a standard string entry. If the text box required a dimension, the user was required to type 2.25 to indicate the size in master units plus some fraction.

MDL Equivalent

By contrast, our MDL interfaces were able to take advantage of the MDL textbox control which easily matched the “Look and Feel” of MicroStation’s interface. MDL textboxes could easily match the coordinate readout setting for instance.

I suppose if that were the only difference between MDL and VBA, the advantage of including VBA into MicroStation would be lost. Fortunately though, VBA’s interface development capabilities are still far superior despite some obvious missing features.

Our abilities have evolved a great deal since last September. We have learned how to control textboxes, and we have developed several tools that make VBA operate a little more like MDL.

Primary Goals

One of our primary goals in building an application is to have it find all of its resources from the parent application. We do this to ensure that all of the tools required by our application have been delivered.

Our attitude is that if, for example, MicroStation works then our software must work. For this reason, we tend to stick to controls delivered with MicroStation VBA.

The color picker falls into this category. According to at least one post on the VBA news group, Bentley intends to deliver new controls later this year. Not wanting to wait we decided to build our own.

Initially, we experimented with ActiveX controls and found that while they seemed to work well in Windows 2000, Windows XP exhibited some conflicts. So while we wait for Bentley to deliver a color picker control we need an alternative. Here’s what we did.

Color Picker Example

We chose to implement a color picker on a user form. A request for color loads the user form and selecting the color unloads the form. Seems fairly simple right?

Building your own tools for VBA is not always intuitive especially where a matrix of controls is required. You see VBA doesn’t provide control arrays. Ok, you may ask, what are these?

Let’s say you want to build a tool box and you have 10 buttons. In VBA, you would insert a user form and place 10 buttons on the form. Each would have a unique name like cmdButton1, cmdButton2, and so on. Each one of these buttons would also have its own unique code block that executes the same command.

```
Private Sub cmdButton1_Click()  
    buttonClicked 1  
End Sub  
Private Sub cmdButton2_Click()  
    buttonClicked 2  
End Sub  
Private Sub cmdButton3_Click()  
    buttonClicked 3  
End Sub  
Private Sub cmdButton4_Click()  
    buttonClicked 4  
End Sub  
Private Sub cmdButton5_Click()  
    buttonClicked 5  
End Sub  
Private Sub cmdButton6_Click()  
    buttonClicked 6
```

```

End Sub
Private Sub cmdButton7_Click()
    buttonClicked 7
End Sub
Private Sub cmdButton8_Click()
    buttonClicked 8
End Sub
Private Sub cmdButton9_Click()
    buttonClicked 9
End Sub
Private Sub cmdButton10_Click()
    buttonClicked 10
End Sub

```

The availability of a control array would allow you to place the 10 buttons but reference them with an array index. You still have 10 buttons but all have the same variable name but the index changes for each button, cmdButton(1), cmdButton(2) and so on. The code listing is reduced from 30 lines to 3 making editing, and bug fixes more manageable.

```

Private Sub cmdButton_Click(index as Integer)
    buttonClicked(index)
End Sub

```

The problem is that there are no control arrays in VBA. Now the color picker example has 256 controls not just 10. And, that's too many lines of unnecessary code for me.

Solving the problem required some help. I found that help in an Excel VBA example posted by Andrew Baker on www.vbusers.com. Mr. Baker created a class module and user form example to implement control arrays in Excel VBA. His example intercepts control events, such as clicks, and textbox changes making it ideal for our color picker.

The Solution

The final solution has four components, two forms, a module, and a class module. In our example, we have stripped out code that doesn't apply to MicroStation or the example to create a streamlined version which is easier to understand.

Forms

The first user form is for the color picker and the other form will launch the color picker. You can think of the second form as the preferences box. To get started, insert two new user forms and name them formCP, for the color picker, and formPref for the preferences example. Make sure that both of these forms have the ShowModal property set to False.

TIPS IN HERE

The color picker form doesn't require any controls. All of its controls will be created programmatically. We will return to the color picker code in a moment.

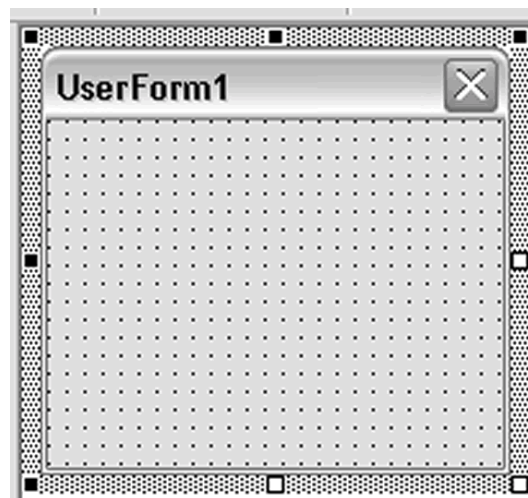


Figure 1 UserForm

Add a label control called lblCPTTest to the preferences form. The label will capture the color number and the color selected. You may also change the caption of the preferences form to something like "Color Picker Test". See Figure 2.

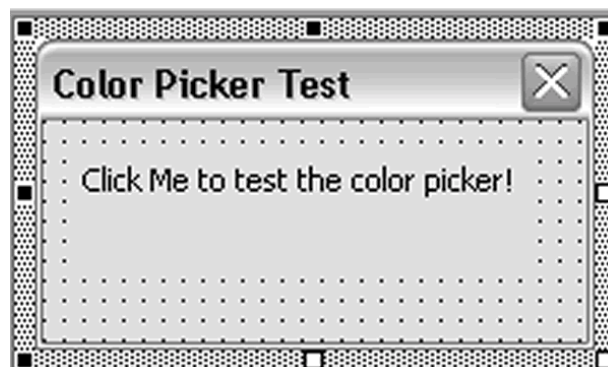


Figure 2 Color Picker Test

Double click on the label and add the following code block to the preferences form.

```

Option Explicit
Private Sub lblCPTTest_Click()
    Load formcp
    formcp.Show
End Sub

```

The code block simply loads and displays the color picker form when the label is clicked.

The color picker requires three code blocks. The first creates and displays the image boxes which make up the color table. The second cleans up the control array when the form is unloaded, and the third is a subroutine that converts the color selected to RGB (red, green, blue) components. The userForm_initialize block is long, but not complicated.

Option Explicit

```
Private oControlArray()As classControlArray
```

```
Private Sub UserForm_Initialize()  
    Dim ArrayRGBLongs() As Long  
    Dim ct As ColorTable  
    Dim r As Byte, g As Byte, b As Byte  
    Dim NewImage As Image
```

MicroStation allows us to extract the attached color table. From this table we can get an array of colors which we store in ArrayRGBLongs.

```
Set ct = Application.ActiveDesignFile.  
        ExtractColorTable  
ArrayRGBLongs = ct.GetColors
```

The object "Me" refers to the user form. In this case we are predefining the height and width of the user form before populating it with image boxes.

```
Me.Height = 192  
Me.Width = 174  
  
Dim ThisBox As Long  
  
ReDim oControlArray(0 To 255)
```

We can define up to 255 colors in our color table. Our color table also allows us to select a "By Level" option.

```
For ThisBox = 0 To 254
```

Extract RGB returns the red, green and blue integers for our color index. VBA controls understand RGB but not MicroStation's color index.

```
ExtractRGB ArrayRGBLongs(ThisBox),r,g,b
```

An image box is created then its background color, size and position are defined.

```
Set NewImage = Me.Controls.Add  
                ("Forms.Image.1")  
NewImage.BackColor = RGB(r, g, b)  
NewImage.Width = 10.5  
NewImage.Height = 10.5
```

The following if construct creates a matrix of image boxes in a similar pattern found on MicroStation's color picker.

```
If ThisBox >= 0 And ThisBox <16 Then  
    NewImage.Left = (10.5 * (ThisBox))  
    NewImage.Top = 0  
ElseIf ThisBox >= 16 And ThisBox <32 Then  
    NewImage.Left = (10.5 * (ThisBox - 16))  
    NewImage.Top = 10.5  
ElseIf ThisBox >= 32 And ThisBox <48 Then  
    NewImage.Left = (10.5 * (ThisBox - 32))  
    NewImage.Top = 21  
ElseIf ThisBox >= 48 And ThisBox <64 Then  
    NewImage.Left = (10.5 * (ThisBox - 48))  
    NewImage.Top = 31.5
```

```
ElseIf ThisBox >= 64 And ThisBox <80 Then  
    NewImage.Left = (10.5 * (ThisBox - 64))  
    NewImage.Top = 42  
ElseIf ThisBox >= 80 And ThisBox <96 Then  
    NewImage.Left = (10.5 * (ThisBox - 80))  
    NewImage.Top = 52.5  
ElseIf ThisBox >= 96 And ThisBox <112 Then  
    NewImage.Left = (10.5 * (ThisBox - 96))  
    NewImage.Top = 63  
ElseIf ThisBox >= 112 And ThisBox <128 Then  
    NewImage.Left = (10.5 * (ThisBox - 112))  
    NewImage.Top = 73.5  
ElseIf ThisBox >= 128 And ThisBox <144 Then  
    NewImage.Left = (10.5 * (ThisBox - 128))  
    NewImage.Top = 84  
ElseIf ThisBox >= 144 And ThisBox < 160 Then  
    NewImage.Left = (10.5 * (ThisBox - 144))  
    NewImage.Top = 94.5  
ElseIf ThisBox >= 160 And ThisBox <176 Then  
    NewImage.Left = (10.5 * (ThisBox - 160))  
    NewImage.Top = 105  
ElseIf ThisBox >= 176 And ThisBox <192 Then  
    NewImage.Left = (10.5 * (ThisBox - 176))  
    NewImage.Top = 115.5  
ElseIf ThisBox >= 192 And ThisBox <208 Then  
    NewImage.Left = (10.5 * (ThisBox - 192))  
    NewImage.Top = 126  
ElseIf ThisBox >= 208 And ThisBox <224 Then  
    NewImage.Left = (10.5 * (ThisBox - 208))  
    NewImage.Top = 136.5  
ElseIf ThisBox >= 224 And ThisBox <240 Then  
    NewImage.Left = (10.5 * (ThisBox - 224))  
    NewImage.Top = 147  
ElseIf ThisBox >= 240 And ThisBox <255 Then  
    NewImage.Left = (10.5 * (ThisBox - 240))  
    NewImage.Top = 157.5  
End If
```

The image box is now added to our control array. We'll learn more about the class module in a moment, but for now all we are doing is indexing the control so that it can be referenced later.

```
Set oControlArray(ThisBox)  
                = New classControlArray  
oControlArray(ThisBox).Initialize  
                NewImage, ThisBox
```

Next

Add the By Level image box.

```
'ADD BYLEVEL OPTION  
Set NewImage =  
    Me.Controls.Add("Forms.Image.1")  
NewImage.BackColor = vbButtonFace  
NewImage.Width = 10.5  
NewImage.Height = 10.5  
NewImage.Left = (10.5 * (15))  
NewImage.Top = 157.5  
Set oControlArray(255) =  
    New classControlArray  
oControlArray(255).Initialize  
    NewImage, 255
```

End Sub

When the color picker is closed it is good practice to free up the memory used by the control array. The following terminate procedure handles the clean up for us.

```
Private Sub UserForm_Terminate()
    Dim ThisBox As Long, indx As Integer
    Dim itmp As Integer
    For ThisBox = 0 To 255
        Set oControlArray(ThisBox) = Nothing
    Next
End Sub
```

Include ExtractRGB in the user form to hide it from the user. Putting the procedure in a class module will accomplish the same thing.

```
Private Sub ExtractRGB(ByVal longColor As
    Long, intRed As Byte, intGreen
    As Byte, intBlue As Byte)
    Dim lngColor As Long

    lngColor = longColor
    intRed = lngColor Mod &H100
    lngColor = lngColor \ &H100
    intGreen = lngColor Mod &H100
    lngColor = lngColor \ &H100
    intBlue = lngColor Mod &H100
End Sub
```

Class Module

The color picker form code declares and then defines an array of controls using the object oControlArray. Each color image is assigned to a container in this array.

When the user clicks on a color image box the form is dismissed and the label in the preferences dialog is set to that color. Further, a mouse over of each image changes the caption in the color picker dialog to indicate the current color.

The classControlArray is an object that we created. It has a method and a property just like any other object might have. Due to the nature of the problem we are attempting to solve however, the class module depends on the programmer to define two sub routines somewhere in the project code to handle the click and mouse over events.

For a complete discussion of class modules “Visual Basic Language Developer’s Handbook” by Ken Getz and Mike Gilbert, SYBEX is an excellent resource. (ISBN: 0-7821-2162-4).

WithEvents tells VBA to assign the same events to the variable as those included in the variable type. In the following example, zoImage inherits the same events as image boxes. This is what causes the click and mouse move events to work.

```
Option Explicit
Private WithEvents zoImage As MSForms.Image
```

zlIndex is a property of the class module. This property is referenced by the click and mouse move events.

```
Private zlIndex As Long
Private Sub zoImage_Click()
    zo_setColor zlIndex
End Sub

Private Sub zoImage_MouseMove(ByVal Button
    As Integer, ByVal Shift As Integer,
    ByVal X As Single, ByVal Y As Single)
    zo_displayColor zlIndex
End Sub
```

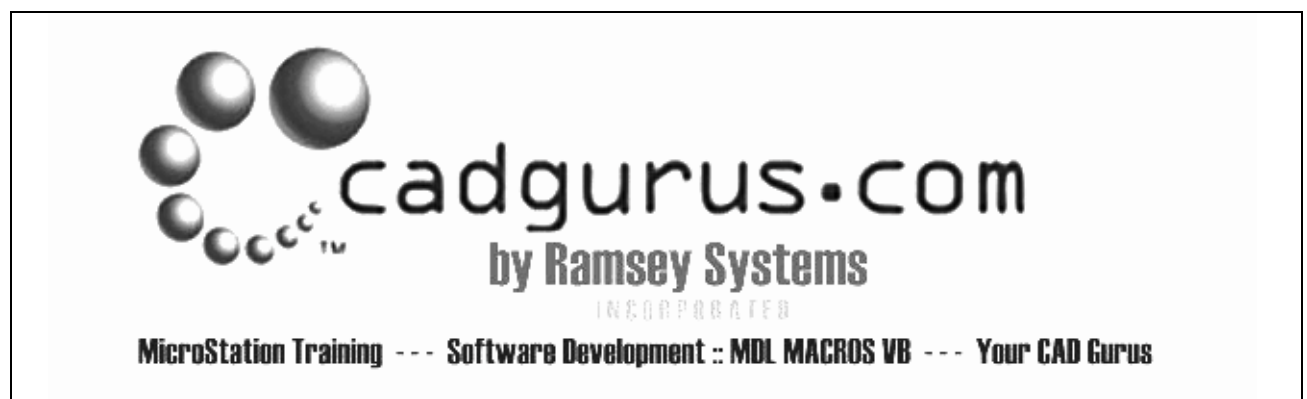
Good programming practice requires that we clean up objects no longer in use. Setting zoImage to nothing releases memory for some other process.

```
Private Sub Class_Terminate()
    Set zoImage = Nothing
End Sub
```

The Initialize method is called from the initialize event of the color picker user form. All it does is set the control’s index and assign zoImage to the control just created by the user form.

```
Sub Initialize(oControl As Object,
    lControlIndex As Long)
    zlIndex = lControlIndex
    Select Case TypeName(oControl)
    Case "Image"
        Set zoImage = oControl
    Case Else
        Stop 'Unrecognized type!
    End Select
End Sub
```

```
Property Get index() As Long
    index = zlIndex
End Property
```



Module

The module contains four subroutines. The first is used to open the test user form. The next two routines `zo_setColor` and `zo_displayColor` are the routines that the class module calls during click and mouse move events respectively.

The last routine is `ExtractRGB` which we saw earlier. It is set to *private* to hide it from the end user.

Normally we would put this into a common class module but to simplify the discussion we duplicated it here.

```
Option Explicit
```

```
Public Sub cpTest()  
    Load formPref  
    formPref.Show  
End Sub
```

`Zo_setColor` is called when the user selects a color from the user form. The first thing it does is unload the form. It then converts color to RGB and sets the preference form label.

```
Public Sub zo_setColor(index As Long)  
    Unload formcp  
  
    Dim ArrayRGBLongs() As Long  
    Dim ct As ColorTable  
    Dim r As Byte, g As Byte, b As Byte  
  
    Set ct = Application.ActiveDesign  
        File.ExtractColorTable  
        ArrayRGBLongs = ct.GetColors  
  
    If index < 255 Then  
        ExtractRGB ArrayRGBLongs(index), r, g, b  
    End If  
  
    If index < 255 Then  
        formPref.lblCPTest.BackColor=RGB(r,g,b)  
        formPref.lblCPTest.Caption="You selected  
            color" & Str(index) & "."  
    Else  
        formPref.lblCPTest.BackColor =  
            vbButtonFace  
        formPref.lblCPTest.Caption="You selected  
            to set color By Level."  
    End If  
End Sub
```

As the user passes over each image on the color picker we use `zo_displayColor` to change the caption of the color picker form.

```
Public Sub zo_displayColor(index As Long)  
    If index < 255 Then  
        formcp.Caption = "Choose Color:"  
            & Str(index)  
    Else  
        formcp.Caption = "Choose Color:By Level"  
    End If  
End Sub
```

```
Private Sub ExtractRGB(ByVal longColor As  
    Long, intRed As Byte, intGreen As  
    Byte, intBlue As Byte)  
    Dim lngColor As Long  
    lngColor = longColor  
    intRed = lngColor Mod &H100  
    lngColor = lngColor \ &H100  
    intGreen = lngColor Mod &H100  
    lngColor = lngColor \ &H100  
    intBlue = lngColor Mod &H100  
End Sub
```

Now try it.

- Click on the Preferences form color label. The color picker form opens.
- Choose a color, the form closes and the label caption changes. The background color of the label is also changed to match the color selected.

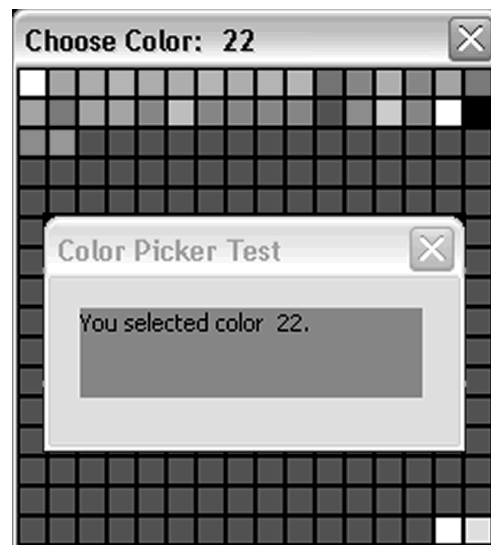


Figure 3 The Completed Color Picker

Go ahead create more sophisticated interface features for MicroStation. You don't have to know how to make ActiveX controls, just use what's delivered.

About The Author

Mark Stefanchuk is a partner with Ramsey Systems, Inc., the developers of cadgurus.com. Mark can be contacted by email on mark@cadgurus.com.

Please email Mark with any feedback or suggestions for future articles.

CAD

Tips in Here