

## Using Visual Basic in Arc8 Raster Processing Form Example

Matt Gregory and Michael Guzy

This is a VERY simplistic introduction to customizing Arc8 with VB (or VBA) partly because I don't fully understand what's going on and partly because it seems like it's going to be a steep learning curve if you're not used to VB or COM programming.

We're going to be using one of Arc8's sample forms to play around with some grid processing and visualization. It uses a form with many different control objects to create this functionality. This should build upon Deanna's presentation of using the various controls and changing their associated properties. Now, we will put the VB code "behind" these controls so that the events generated by clicking buttons, sliding bars, etc. do something.

The best reference I found for all of this stuff was under the Help->Help for Developers->ArcObjects Developer Help. It also has an online address at <http://arconline.esri.com/arcobjectsonline/>. Here you can read about the Component Object Model (COM) and how ArcObjects and VB fit into the equation. There are also sample bits of code and tools that you can directly put into your projects and the object model diagrams. I didn't try the sample code, but it might be a good thing for someone to try.

The code we are working with can be loaded by doing the following :

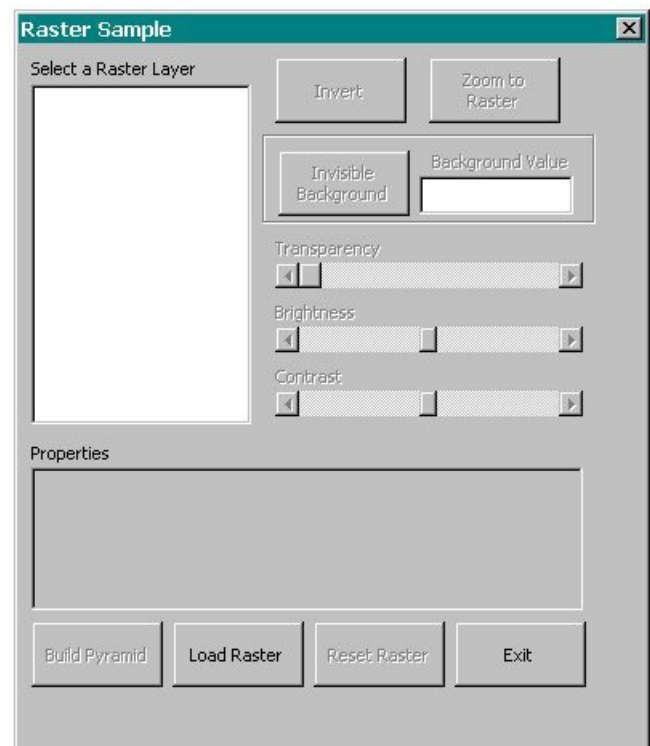
1. Open up a new ArcMap session
2. Go to Tools->Macros->Visual Basic Editor (alternatively Alt-F11)
3. Once the Visual Basic window has opened, go to File->Import File and navigate to C:\arcgis\arcexe81\ArcObjects Developer Kit\Samples\Raster\Simple Raster Sample.
4. Open the form called frmSimpleRasterSample.frm. In the VB Project Explorer window, under Project->Forms, you should see the form called frmSimpleRasterSample.
5. Hit F5 to run the application.

### Running the Simple Raster Application

The menu at right should pop up. Go ahead and play around with the various commands to know what they do. (I wasn't able to load any of the previous rasters we had worked with into the project. VB always blew up. I ended up using the rasters under the ArcTutor directory, especially a DEM of Yellowstone at C:\arcgis\ArcTutor\Catalog\Yellowstone\dem30. Any raster that loads should be fine for this application, though.) Here's what the help says about the various commands:

#### ***Description:***

*This sample demonstrates how to retrieve raster layers from ArcMap, display some raster properties, modify the raster display, build raster pyramids and load additional raster layers. It contains functionality similar to ArcMap's Effects toolbar. The sample is written in VBA and is self contained in one VBA form. The VBA form lists the raster layers in your currently selected map. Clicking on one of these layers will display some raster properties. Toggle buttons and sliders are enabled to reflect the current state of the raster. Listed below are details of the*



*functions available in this sample:*

*Load more raster's by clicking on the Load Raster button. This will use a GxDialog with a GxFilterRasterDatasets to browse for raster datasets and bands, these are then loaded into ArcMap.*

*If the raster does not have any pyramids, the Build Pyramids button will be enabled and clicking on it will give the option to build a pyramids for all bands in a raster. This is done through the IPyramid interface on each raster band. Pyramids help improve the display performance of raster's when viewed at scales much less than the raster's resolution. This is useful for viewing a mosaic of large raster layers.*

*A single raster's display can be modified interactively using sliders (scrollbar controls) to change the raster's transparency, brightness and contrast. This is done through the ILayerEffects interface.*

*If the raster renderer supports stretch, then clicking on Invert will change the raster to appear like a photo negative. This option is a toggle and a second click will remove the inversion. The IRasterStretch interface on the raster renderer is used for this effect.*

*Clicking on Invisible Background will make the raster value in the text box become invisible or see through. This option is also a toggle and a second click will remove the effect. Setting a background raster value to be invisible is useful, for example, when a raster has some black borders which obscure underlying layers. In this case the black color (typically a raster value of 0) can be made invisible. Again this option is only enabled if the raster renderer supports the IRasterStretch interface.*

*Clicking the Zoom to Raster button will change ArcMap's display to the visible extent of the selected raster.*

*Clicking on the Reset Raster button will set transparency, brightness and contrast to 0. Invisible background color will be removed and the raster will not be inverted.*

## **Looking at the VB code and understanding the ArcObjects model**

Once you're done playing around with the different functions of the application, click Exit to return to the Visual Basic window. We'll now look at the Visual Basic code associated with clicking the "Load Raster" button as an example of the complexity of the ArcObjects model. First, click on the "Load Raster" button in the form to see the properties associated with this command button. As pointed out last time, the properties window shows all of the properties of this command button, including size, position, name and caption. To open the code window, either hit F7 or go to View->Code (or double-click the button in the form). On the left-hand dropdown menu, choose "cmbLoadLayer" (the command button associated with the "Load Raster" event), and the right-hand dropdown menu should be "click". This should bring up the subroutine called Private Sub cmbLoadLayer\_Click(). I'm including the full text of that code below in `this font`. The comments from ESRI are preceded with an apostrophe. Where I didn't understand what was going on (that is, almost everywhere), I've added some additional comments and figures.

```
`  
' Loads a raster Layer using GxDialog to browse for a raster  
'  
Private Sub cmbLoadLayer_Click()  
    Dim pGxDialog As IGxDialog  
    Dim pGXSelect As IEnumGxObject  
    Dim pGxObject As IGxObject  
    Dim pGXDataset As IGxDataset  
    Dim pGXDatabase As IGxDatabase  
    Dim pDataset As IDataset  
    Dim pRasterLayer As IRasterLayer
```

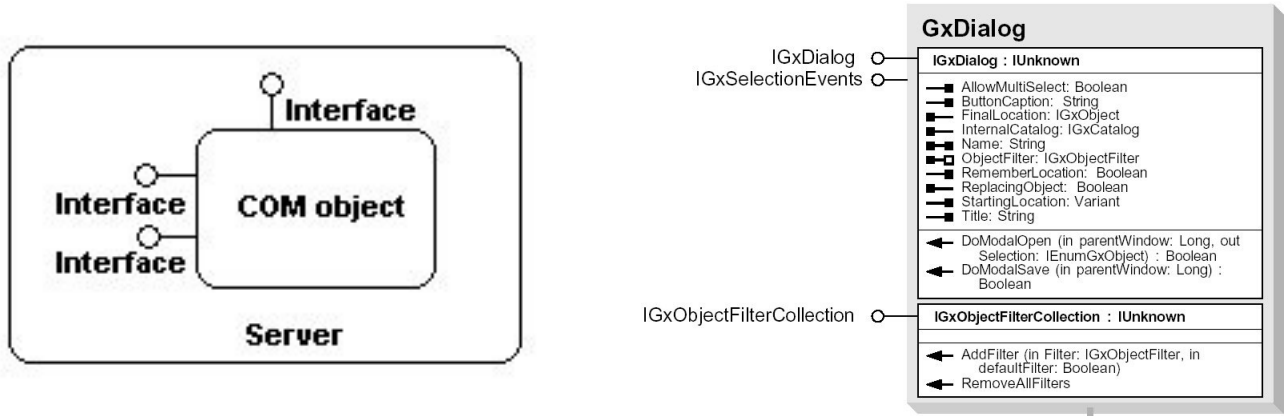
```
Dim pEnumDataset As IEnumDataset
Dim layerAdded As Boolean
```

The Dim statement declares variables and allocates storage space. The above code just declares a bunch of variables that will be used in this subroutine. The Boolean type might look familiar, but that is probably the only one that does. All of the other types (such as IEnumDataset and IGxDialog) are called interfaces, which is one of the key components of COM programming. From the ArcHelp :

*COM interfaces are how COM objects communicate with each other. When working with COM objects, the developer never works with the COM object directly but gains access to the object via one of its interfaces. COM interfaces are designed to be a grouping of logically related functions. The virtual functions are called by the client and implemented by the server.*

*The interface sets out what requests can be made of an object that chooses to implement the interface ... Functionality is modeled abstractly with the interfaces, and implemented within a class implementation. Classes and interfaces are often referred to as the “What” and “How” of COM. The interface defines what an object can do and the class defines how it is done.*

What I understand from that is the the interfaces are a way to establish a connection and talk to the different ArcObjects without having to have direct access to the code of the called object. The operations that we can do to or ask of the objects are limited by what this interface allows. Here’s the conceptual figure of the interface/object relationship and the specific one for the IGxDialog/GxDialog relationship.



```
` Set up filters on the files that will be browsed
Set pGxDialog = New GxDialog
```

Now that we have the interface to the object established (pGxDialog), we create a new instance of the object, in this case a dialog box.

```
pGxDialog.AllowMultiSelect = True
pGxDialog.Title = "Select Raster"
Set pGxDialog.ObjectFilter = New GxFilterRasterDatasets
```

We can also set particular properties of that object through the interface, such as whether or not to allow it select multiple list items and the dialog box caption. The last line says to filter for raster datasets only, using the same interface/object syntax. The ObjectFilter property of the pGxDialog is an interface to a IGxObjectFilter.

```

    ` Bring up the browse dialog
    If (pGxDialog.DoModalOpen(m_pMxDoc.ActiveView.ScreenDisplay.hWnd, pGXSelect) = False)
Then
    ` Exit the function if the user cancels the browse dialog
    Exit Sub
End If

```

The DoModalOpen method activates the browsing window, allows the user to select one or more raster layers and stores the resulting raster layers in the pGXSelect variable. If by chance the user cancels before selecting a raster, the entire subroutine will exit. If not, it continues on . . .

```

    ` Got a valid selection from the GX Dialog,
    ` now extract the raster data sets

    ` loop through the selection enumeration
    layerAdded = False
    pGXSelect.Reset
    Set pGxObject = pGXSelect.Next

```

Reset the pGXSelect object to the beginning of the list and get the first object

```

Do While (Not pGxObject Is Nothing)

    Set pGXDataset = pGxObject
    Set pDataset = pGXDataset.Dataset
    If (TypeOf pGxObject Is IGxDataset) Then

        If (pGXDataset.Type = esriDTRasterDataset) Or (pGXDataset.Type = esriDTRasterBand)
Then

            ` Take the raster dataset or band and create a valid layer from it
            Set pRasterLayer = New RasterLayer
            pRasterLayer.CreateFromDataset pDataset

            ` Add the new layer to the document
            m_pMxDoc.AddLayer pRasterLayer

            layerAdded = True

        End If
    End If

    ` Move onto the next selected raster
    Set pGxObject = pGXSelect.Next
Loop

```

The above loop keeps cycling until it reaches the end of the list (with the test condition being “Not pGxObject Is Nothing”?????). It first makes sure that it has the correct interface to communicate with the GxDataset object and then queries the object to ensure it is a valid raster object. If so, it creates an ArcMap layer in the current document and sets the layerAdded flag to True. The last step is to increment to the next layer selected in the pGXSelect list.

```

If layerAdded Then
    ` Update ArcMap TOC etc.
    m_pMxDoc.UpdateContents

```

If a layer was added, refresh the current TOC in the ArcMap window

```
` Update the list of layers and then the buttons to reflect this
  InitialiseLayers
  UpdateUI
End If

End Sub
```

The routine then calls two other subroutines (InitialiseLayers and Update UI) which are also defined in the form, frmSimpleRasterSample. They are not associated with any specific control event so they will appear when the left-hand pulldown menu is set to “General”. The InitialiseLayers subroutine “populates the list control with raster layer names and updates the layerIndexLookup”. The UpdateUI subroutine updates the user interface by calling other subroutines based on the active layer in the raster sample application. These might be good to explore as well ...

Past this, it’s probably worthwhile to poke around the various subroutines to try to decipher what they do. Also, there is a user form toolbar (under View->Toolbars->UserForm) which allows you to “beautify” various controls on a form.

### **Helpful things :**

- Placing the cursor in properties, methods or class names and hitting F1 in Visual Basic will bring up the ArcObjects help window. This will not work for variable instances, e.g. :

```
Dim pLayer As ILayer (F1 will work on ILayer, but not pLayer)
pLayer.name (F1 will work on name, but not pLayer)
```

- Stepping through the code using the debugger is helpful for figuring out what is happening and in which order. Use View->Toolbars->Debug to show the various debugging options, such as stepping through the code a line at a time and setting up variable watch windows to determine values at run-time.

### **Deep thoughts . . .**

The customization of Arc8 looks to be pretty challenging, especially when most of us are coming from an AML background (this application was designated as easy!). The good news is that the menu interfaces already built into ArcMap and ArcCatalog are probably going to be easier to understand, so that tasks like making maps and simple analyses that we’ve written AMLs for in the past might be easier in Arc8. It seems like we need to identify places where we might need to customize (e.g. running loops) and concentrate some programming efforts there.

In addition to the ESRI ArcObjects web site, the ESRI virtual campus offers a class on learning VB, the first module of which is free. Also, if you’re interested in the whole VB-COM connection, there looks to be a promising tutorial at <http://www.develop.com/tutorials/vbcom/>.