Aaron Marks
Geog 485
Lesson 2

Building a Button, Command and Menu

Summary

The idea behind this lesson was to get into working with various interfaces and how to Query into them as well. This also includes how to maneuver around the VBA help resources such as Object Diagrams as well as the Desktop Help. Knowing how to find what you are looking for is a crucial step in being able to program procedures using interfaces or objects that contain bundles of code we would like to use.

With the first two practice exercises it became clear to me that there is no way around using the Desktop Help past the first few easily memorized interfaces. Even then it isn't easy to memorize what all they are capable of. The Hints about the first property being Boolean and the second being an integer (or count) helped eliminate most of the properties. The tricky part was finding the correct interface. It took me a while to find it but having used a similar interface, in the previous practice problem 9(ILayerEffects)), I found ILayerFields. I was looking for words like Field, count, attributes and knew that it was under an Ilayer type interface, since they're attributes to the layer.

Lesson 2

This lesson not only covered how to go about adding a toolbar, button and menu, but it also covered how to put some code behind what the user clicks on. The click, or command, has its own specific chunk of code that is carried out when clicked on. Different buttons all do something slightly different than to a similar one. In this case each button changes the zoom of the open map, similar in all ways other than the preset zoom values. With the provided code it was easy to fill in the blank following the Desktop Help alone, but the OMD could have helped some as well.

Coding the initial button required the most thought as the blanks were empty. The first two blanks are pretty basic syntax with query interfaces. The next blank I originally plugged in 'extent' here, which works if you want apply the selected zoom to the window as it was left, as opposed to the original. I found **FullExtent** as a property of **IActiveView** as well; this would use the full extent to be considered as no zoom, or 0%. The **pEnv.Expand** method changes the X and Y envelope by entered ratio. The final blank is what actually changes the zoom by making the extent equal to the previously changed envelope.

Overall I felt this went fairly well especially after the practice exercises. To further my learning I could have incorporated a retro-active zoom, meaning each time the user clicks on +125% it zooms in 125% of the active view. That would just require the **Set pEnv = pActiveView.FullExtent** line to be set to **Set pEnv = pActiveView.Extent**. I added a 0% which would be usefull in both situations, and with this I excluded the **pEnv.Expand** line and kept the **Set pEnv = pActiveView.FullExtent** line in order to reset the zoom to 0% or the full extent.

Screen Captures