Symbolize Layers Programmatically

Make renderers and set their properties

By Eulimar Cunha Tiburcio, Denver, Colorado

This article describes how to programmatically symbolize layers through the use of renderers.

Map layers contain properties that define the symbology used to display its features. Users can control a layer's symbology through working with its legend. Programmers can access these legends directly by leveraging renderers and controlling their various properties via code.

When you make a feature layer renderer, you have the ability to create your own symbology. You also can use symbols previously created by others. Using symbols that already exist can save a great deal of programming time. ArcGIS symbols are stored in the Style Manager as styles, style gallery classes, and style gallery items.

Styles contain symbols unique to an industry or discipline and contain style gallery classes that are groups of similar symbols or map elements. Marker symbols are a style gallery class and so are line symbols, colors, and north arrows. Style gallery classes contain style gallery items, which are individual symbols or elements. Within a style gallery class, items can belong to different categories such as marker symbols or fill symbols.

To use symbols from an existing style, follow these three steps:

1. Get a reference to the specific style gallery. This is the object that contains all the styles such as Conservation, Crime Analysis, Environmental, Hazmat, Mining, Petroleum, Survey, Transportation, Water, Wastewater, and Weather.
2. Get an enumeration or list of style gallery items.
3. Get the specific style gallery item.

Although a StyleGallery is composed of many StyleGalleryClasses, you don't have to get a style gallery class. As Figure A shows, you can go straight from the Gallery to an Enumeration (called an "Enum" for short) containing the item. To get the Enum, use the IStyleGallery interface's Items property. This property has arguments that specify both the style and the style gallery class you want. Listing 1 shows the IMxDocument interface's StyleGallery property is used to get the style gallery in Visual Basic for Applications (VBA).

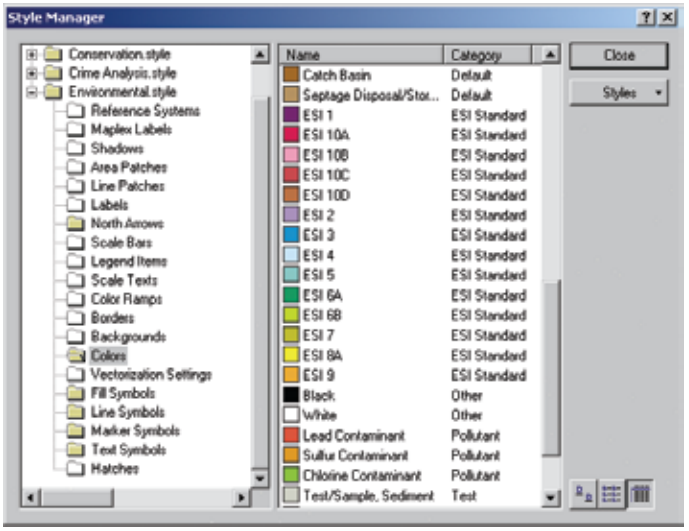
```
Dim pMxDoc As IMxDocument
Set pMxDoc = ThisDocument

Dim pStyleGallery As IStyleGallery
Set pStyleGallery = pMxDoc.StyleGallery
```

Listing 1: Getting the style gallery

The Items property on IStyleGallery returns an Enum of symbols. The Items property has three arguments for specifying the style gallery class, the style, and the symbol category. The code in Listing 2 returns an Enum of line symbols in the Environmental style that belongs to the Pollutant category. Figure B illustrates the list of gallery items (the Enum) returned by this code.

Dialog showing style gallery items



Dialog showing categories

```
Dim pEnumStyleGallery As IEnumStyleGalleryItem
Set pEnumStyleGallery = pStyleGallery.Items _
     ("Line Symbols", "Environmental.style", _
          "Pollutant")
```

Listing 2: Returning an Enum of line symbols

To navigate through the Enum, use the Next and Reset methods on the EnumStyleGalleryItem coclass. When you first get the Enum, a pointer is placed at the top of the list (before the first symbol). You'll need to move the pointer to begin cycling through the list. The Enum's Next method moves the pointer down one symbol in the list, returning that symbol's IStyleGalleryItem interface (see Listing 3).

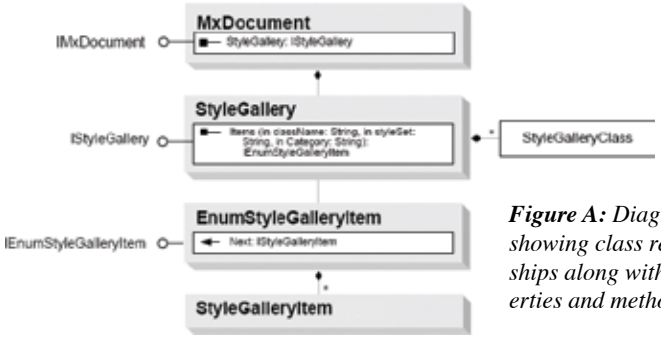


Figure A: *Diagram showing class relationships along with properties and methods*

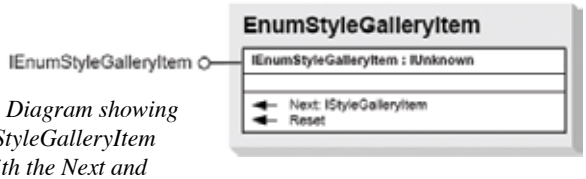


Figure C: *Diagram showing the EnumStyleGalleryItem coclass with the Next and Reset methods*

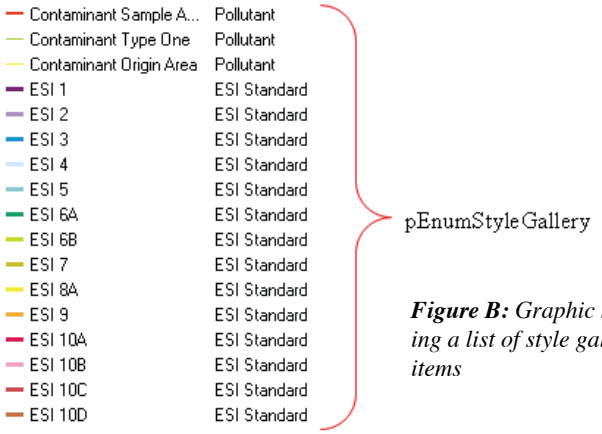


Figure B: *Graphic showing a list of style gallery items*

```
Dim pStyleItem as IStyleGalleryItem
Set pStyleItem =  pEnumStyleGallery.Next
```

Listing 3: Moving the point down one symbol

When you run the Next method on the last symbol in the Enum, the pointer drops off the list and the pointer will return a null value. If you need to cycle back through the list, you can move the pointer back to the top by running the Reset method.

The easiest way to move through the Enum is to put the Next method inside a Do Until loop. The loop below stops when the pointer is at the bottom of the Enum, pointing at Nothing.

The IStyleGalleryItem interface has a Name property, which gives you a way to test each item in the Enum. An item's name is the name that appears in the Style Manager as shown in Figure D. For example, if you want the Contaminant Origin Area line symbol, inside the loop, you could use an If Then statement to test each item for that name.

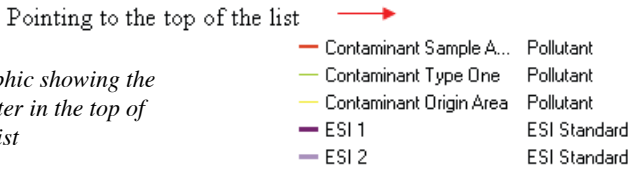
Once you find the symbol, you get it (that is, you get one of its interfaces) using IStyleGalleryItem's Item property. Most likely, you have plans for this symbol, such as adding it to a renderer, that require the ILineSymbol interface. The Item property, however, returns IUnknown.

Every ArcObjects class has IUnknown. The Item property takes advantage of this fact and is able to return such a wide variety of symbols (line symbols, marker symbols, fill symbols, colors, and color ramps) because it returns their IUnknown interface.

To get an interface to the Contaminant Origin Area symbol, you could declare a variable to IUnknown, set it with the Item property, declare an ILineSymbol variable, and set it by doing QueryInterface from IUnknown to ILineSymbol. Or, more conveniently, you could let VBA do the QueryInterface for you with the code in Listing 4. You just declare the variable to ILineSymbol and set it with the Item property.

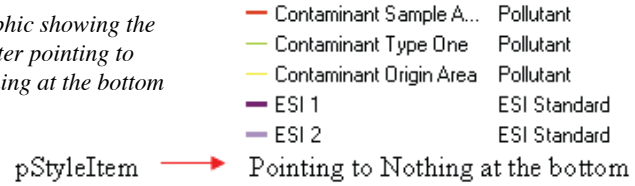
Continued on page 56

Graphic showing the pointer in the top of the list



Graphic showing the pointer pointing to the first item on the list

Graphic showing the pointer pointing to Nothing at the bottom



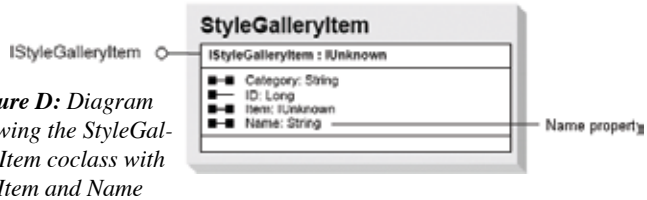


Figure D: *Diagram showing the StyleGalleryItem coclass with the Item and Name properties*

Symbolize Layers Programmatically

Continued from page 55

```
Dim pLineSymbol As ILineSymbol

Do Until pStyleItem Is Nothing
  If pStyleItem.Name = "Contaminant Origin_
            Area" Then
     Set pLineSymbol = pStyleItem.Item
  End If
  Set pStyleItem = pEnumStyleGallery.Next
Loop
```

Listing 4: Putting the Next method in a Do Until loop to catch the item and QI it to an ILineSymbol Interface

After getting the symbol, you can assign it to a layer renderer.

The diagram in Figure E shows that FeatureRenderer is connected to FeatureLayer with a general association. This means that each FeatureLayer has a FeatureRenderer. The FeatureRenderer abstract class has eight subclasses, representing different legend types. A SimpleRenderer (see Listing 5) draws all features in one symbol and one color.

```
Dim pRenderer As ISimpleRenderer
Set pRenderer = New SimpleRenderer
```

Listing 5: A simple renderer that draws all features with one symbol and one color

After creating a renderer, you set its Symbol and Label properties. The Symbol property is a byRef property (open barbell), so you'll need to use the Set keyword. (Refer to Figure F). Next, (shown in Listing 6),

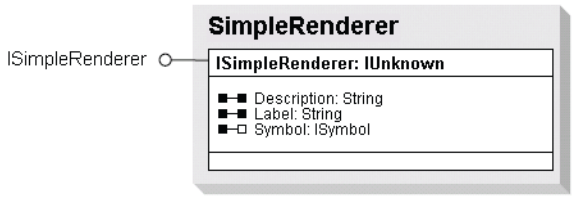


***Figure F**: Diagram showing properties in the SimpleRenderer coclass*

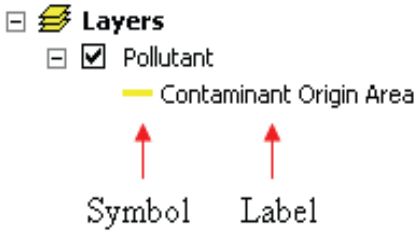


***Figure G:** Graphic showing the Symbol and Label properties of the SimpleRenderer coclass in the ArcMap table of contents*

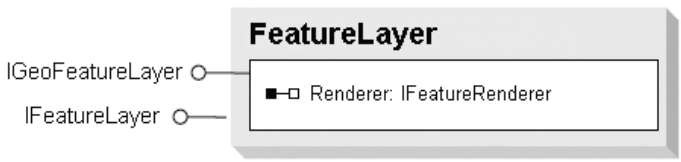


Diagram showing the Renderer property of the FeatureLayer coclass

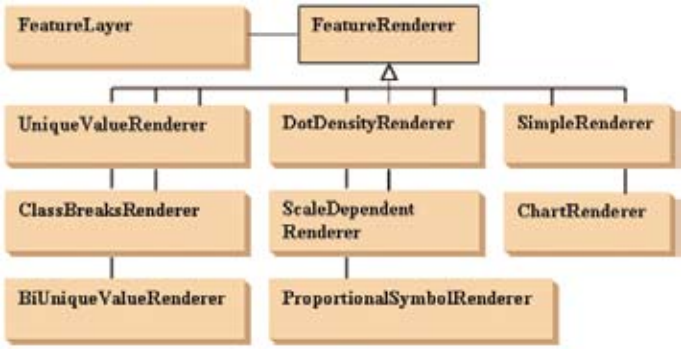


***Figure E:** Diagram showing the class that inherits from the FeatureRenderer abstract class*

set the renderer's Symbol property to the Contaminant Origin Area line symbol and its Label property to Contaminant Origin Area. The Label property sets the text that goes with the symbol in the ArcMap table of contents as shown in Figure G.

```
Set pRenderer.Symbol = pLineSymbol
pRenderer.Label = "Contaminant Origin Area"
```

Listing 6: Setting Symbol and Label properties

The renderer has been given a symbol and a label, but it must still be associated with a feature layer. To make this association, you set the Renderer property on the FeatureLayer class's IGeoFeatureLayer interface. If you haven't already created the feature layer, make one with the code in Listing 7.

```
Dim pGFeatureLayer As IGeoFeatureLayer
Set pGFeatureLayer = New FeatureLayer
```

Listing 7: Making a feature layer

If the layer already exists, and you have a variable pointing to IFeatureLayer, QueryInterface to IGeoFeatureLayer. The Renderer property is also a byRef property that requires the Set keyword. Before you can see the results of your work, you have to refresh the map display and update the table of contents using the code in Listing 8.

```
pMxDoc.ActiveView.Refresh
pMxDoc.UpdateContents
```

Listing 8: Refreshing the map display and updating the table of contents

For more information on this tutorial, contact
Eulimar Cunha Tiburcio, GIS Developer
Denver, Colorado
E-mail: eulimar@lycos.com

About the Author

Eulimar Cunha Tiburcio received a bachelor's degree in engineering mechanics, a master's degree in civil engineering, and a doctorate in civil engineering from Federal University of Ceara in Brazil. He combines expertise in graphic design, cartography, GIS, and geovisualization to provide solutions for water resources projects. He also has developed scripts with Visual Basic and ArcObjects for the ESRI Support Center (arcscripts.esri.com).