Exploring ArcObjects

By Jeff Jackson, ESRI Development Lead

Editor's Note: *With the release of ArcInfo 8, GIS developers can use a new COM-based development technology, ArcObjects, that provides a well-documented data model with embeddable components. With ArcObjects, developers can customize and extend ArcInfo at the lowest level or construct new GIS applications. However, derivative applications require an ArcInfo license. ArcMap and ArcCatalog, two new applications available with the release of ArcInfo 8, are COM based. The author examines the workings of the ArcMap object model to illustrate some of the basic concepts you need to understand so that you can use ArcObjects. Because Visual Basic for Applications (VBA) is the development environment that comes with ArcInfo 8, the examples in the article will be in Visual Basic. However, ArcInfo 8 can be customized with any COM-compliant programming language such as Visual C++. This article assumes that you are conversant with ArcInfo, familiar with object-oriented programming, and have some basic knowledge of COM.*

ArcObjects is the collection of COM components that provides the underpinning for two new ArcInfo 8 applications—ArcMap and ArcCatalog. This set of components includes more than 1,200 objects that may be used to customize, extend, or construct GIS applications. Initially, an object model of this size can seem a little overwhelming. Taking a close look at the object model for ArcMap will familiarize you with some of the fundamental components in the larger object model.

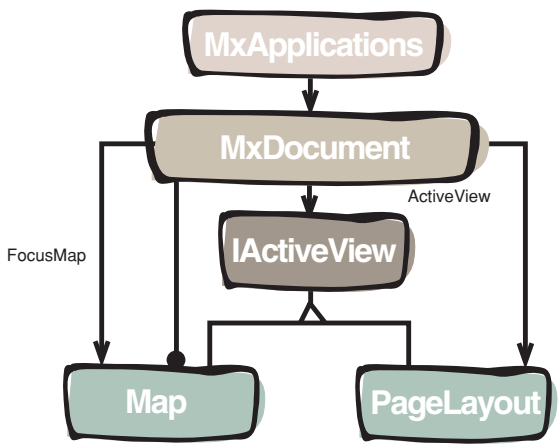

Figure 1

The Basic Structure

Figure 1 shows a conceptual object model for ArcMap. The top-level object, MxApplication, represents ArcMap itself. MxApplication manages a single document, MxDocument. ArcMap uses a Single Document Interface (SDI) in contrast to ArcView GIS, which uses a Multiple Document Interface (MDI). MxDocument manages a collection of Map objects and a PageLayout object. When ArcMap is in geographic view, the ActiveView is a Map. When it's in layout view, the ActiveView is the PageLayout. Regardless of the type of view, there is always a single FocusMap.

If ArcMap displays a document with three data frames in layout view, there will be three Maps attached to the MxDocument, the ActiveView property will return the PageLayout, and the FocusMap will return the Map corresponding to the data frame that is highlighted. The VBA script listed in Figure 2 illustrates this part of the object model. The script changes the ActiveView's extent and zooms in by 75 percent.

This script is fairly straightforward. The real work is getting the current extent, which is an IEnvelope interface, from the ActiveView, shrinking it, and making the modified envelope the new extent. This script operates on the ActiveView through the IActiveView interface. The ActiveView can be either the PageLayout or a Map depending on what the end user is looking at. This script will function correctly in either case.

Figure 2

```
Sub Zoom()
  ' get the active view
  Dim mxDoc As IMxDocument
  Set mxDoc = Application.Document

  Dim activeView As IActiveView
  Set activeView = mxDoc.activeView

  ' get the active view's extent
  Dim ext As IEnvelope
  Set ext = activeView.Extent

  ' shrink the extent
  ext.Expand 0.75, 0.75, True

  ' set the extent
  activeView.Extent = ext
  activeView.Refresh
End Sub
```

Working with Interfaces

In some cases you will want to work specifically with either the Map or the PageLayout through a different interface. In this situation, a COM-object model differs greatly from a traditional object-oriented model like Avenue or MapObjects. An interface is a collection of related methods and properties. COM objects expose multiple interfaces. For example, the Map exposes not only the IActiveView interface but the IMap interface and several others.

Figure 3 shows the Set statement being used to acquire the IMap interface from the activeView variable. Note that this script uses the TypeOf statement to handle the situation in which the ActiveView is not a Map and consequently does not expose the IMap interface. If the TypeOf statement was not used, running the script when the ActiveView is a PageLayout would result in a type mismatch error when the Set statement was encountered.

Figure 3

```
Sub ClearLayers()
  ' get the active view
  Dim mxDoc As IMxDocument
  Set mxDoc = Application.Document

  Dim activeView As IActiveView
  Set activeView = mxDoc.activeView

  If TypeOf activeView Is IMap Then
    Dim map As IMap
    Set map = activeView  ' acquire the IMap interface

    map.ClearLayers       ' remove the layers
    mxDoc.UpdateContents  ' refresh the table of contents
    activeView.Refresh    ' refresh the display
  End If
End Sub
```

Looking at ScreenDisplay

Both the PageLayout and the Map have an associated ScreenDisplay object that is used for all graphic rendering. ScreenDisplay maintains a DisplayTransformation. You can access the properties of DisplayTransformation such as VisibleBounds and use the methods to convert coordinates and distances between map and device units. This is illustrated by the diagram in Figure 4.

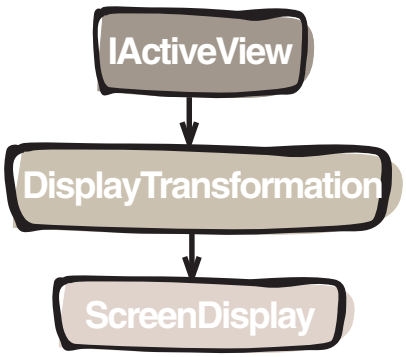

Figure 4

Events such as adding a new layer to the map will cause the ScreenDisplay to trigger a redraw. The drawing process occurs in three phases. The drawing order is the geography, the selection, and the annotation. A drawing cache is an off-screen bitmap of the rendered map that is stamped onto the screen whenever the window is painted. ScreenDisplay manages several drawing caches. When you initiate drawing to a ScreenDisplay you must specify the destination cache. If the constant esriNoScreenCache is specified, the graphics will draw directly on the ScreenDisplay's window and will not be cached. If another window obscures the display, the graphics will be lost. The following script draws the text "Hello" in the center of the ScreenDisplay using the esriNoScreenCache constant.

Figure 5

```
Private Sub Hello()
  Dim mxDoc As IMxDocument
  Set mxDoc = Application.Document

  Dim activeView As IActiveView
  Set activeView = mxDoc.activeView

  ' set up a text symbol to draw with
  Dim sym As ITextSymbol
  Set sym = New TextSymbol

  ' change the font size to 18
  Dim fnt As IFontDisp
  Set fnt = sym.Font
  fnt.Size = 18
  sym.Font = fnt

  ' draw into the active view's display
  With activeView.ScreenDisplay
    .StartDrawing .hDC, esriNoScreenCache
    .SetSymbol sym

    Dim bnds As IArea
    Set bnds =
.DisplayTransformation.VisibleBounds
    .DrawText bnds.Centroid, "Hello"

    .FinishDrawing
  End With
End Sub
```

In the example in Figure 5, note that the drawing calls SetSymbol and DrawText are bracketed within the StartDrawing and FinishDrawing methods.

Working with Layers

Maps maintain a collection of layers. A layer is any object that implements the ILayer interface. Examples of layer objects include FeatureLayer, GroupLayer, GraphicsLayer, AnnotationLayer, CadLayer, TinLayer, and RasterLayer. The Layer(index) and LayerCount properties of the IMap interface can be used to navigate the layer collection. The function illustrated in Figure 6 finds and returns the layer with the specified name.

Figure 6

```
Function FindLayer(map As IMap, name As String)
As ILayer
  Dim i As Integer
  For i = 0 To map.LayerCount - 1
    If map.Layer(i).name = name Then
      Set FindLayer = map.Layer(i)
      Exit Function
    End If
  Next
End Function
```

A FeatureLayer manages a connection to a vector data source such as a coverage, shapefile, or Spatial Database Engine (SDE) layer through a FeatureClass object. The script in Figure 7 obtains a FeatureClass object for a shapefile by opening the Workspace that contains the shapefile using ShapefileWorkspaceFactory. It then creates a FeatureLayer, attaches the FeatureClass to the layer, and adds it to the FocusMap.

Figure 7

```
Sub AddLayer()
  ' use a workspaceFactory to open the workspace
  Dim wksFact As IWorkspaceFactory
  Set wksFact = New ShapefileWorkspaceFactory

  Dim wks As IFeatureWorkspace
  Set wks = wksFact.OpenFromFile("c:\Data\shp", 0)

  ' open the featureClass
  Dim fc As IFeatureClass
  Set fc = wks.OpenFeatureClass("BigCypress")

  ' create a featureLayer
  Dim lyr As IFeatureLayer
  Set lyr = New FeatureLayer
  Set lyr.FeatureClass = fc

  ' name the layer with the featureClass name
  Dim ds As IDataset
  Set ds = fc
  lyr.Name = ds.Name

  ' add the layer to the map
  Dim mxDoc As IMxDocument
  Set mxDoc = Application.Document

  Dim map As IMap
  Set map = mxDoc.FocusMap
  map.AddLayer lyr

End Sub
```

Continued on page 30

Exploring ArcObjects

Continued from page 29

FeatureLayers manage a feature selection and expose the IFeatureSelection interface to manipulate the selection. The script in Figure 8 makes a selection using a logical query with a SQL WHERE clause.

Figure 8

```
Sub SelectFeatures()
  Dim mxDoc As IMxDocument
  Set mxDoc = Application.Document

  ' find the layer we're going to select from
  Dim lyr As IFeatureLayer
  Set lyr = FindLayer(mxDoc.FocusMap, "BigCypress")

  ' get the layer's selection interface
  Dim sel As IFeatureSelection
  Set sel = lyr

' create a query filter
  Dim filter As IQueryFilter
  Set filter = New QueryFilter

  ' set up the where clause and the spatial reference
  filter.WhereClause = "NAME = 'Hampton'"

  Dim shapeField As String
  shapeField = lyr.FeatureClass.ShapeFieldName
  Set filter.OutputSpatialReference(shapeField) =_
  mxDoc.FocusMap.SpatialReference

  ' invalidate the current selection on the display
  mxDoc.ActiveView.PartialRefresh_
  esriViewGeoSelection, Nothing, Nothing

  ' select features and invalidate the new selection
  sel.SelectFeatures filter,_
  esriSelectionResultNew, False
  mxDoc.ActiveView.PartialRefresh_
  esriViewGeoSelection, Nothing, Nothing

  ' notify everyone that we've modified the selection
  Dim selEvents As ISelectionEvents
  Set selEvents = mxDoc.FocusMap
  selEvents.SelectionChanged
End Sub
```

The SelectFeatures method on the IFeatureSelection interface does all the work in this script. A QueryFilter object defines the criteria for the selection. The WhereClause property is set to a SQL expression and the OutputSpatialReference property is set to control how the features are projected on the fly as they are retrieved from the database. A SpatialFilter object could have been used instead of a QueryFilter in order to specify a spatial constraint.

The calls to PartialRefresh on the map through the IActiveView interface take care of redrawing the display so that the old selection is erased and the new selection is drawn. The final three lines of the script call the SelectionChanged method on the map through the ISelectionEvents interface. This method notifies any components in the system that have registered themselves with the map that the selection has changed. The Attribute Table Window listens for selection events so that it can synchronize the highlighting of its rows to match the selection. If the script in Figure 8 did not call the SelectionChanged method, the Table Window would not reflect the selection in the active view. This form of communication, called working with events, is used throughout ArcObjects.

Responding to Events

Up until now we've only considered manipulating ArcObjects using scripts that start with the top level object and navigate to the object that will be manipulated. An interesting aspect of ArcObjects is the ability to wire into the object model and attach code that will be executed whenever something noteworthy happens such as the map's selection changing. Many parts of the object model fire events. COM provides a framework for listening and responding to events.

If you are using VBA or Visual Basic, most of the plumbing required to respond to events is done for you behind the scenes. You just have to write a script for each event you are interested in and a script to start listening for events. The example shown in Figure 9 illustrates how to work with events in the VBA environment. The following code declares a global variable using the WithEvents statement. WithEvents tells the development environment that the object variable will be used to respond to the object's events.

```
Dim WithEvents g_Map  As map
```

After the variable is declared, the variable can be chosen from the Object drop-down list in the VBA interface. Clicking the Procedure drop-down list will show the various events that are exposed by the Map. This example uses the SelectionChanged method to attach code to that event. The script in Figure 9 responds to the SelectionChanged event that is fired by the map whenever the selection changes and populates a ListBox in a UserForm with the names of the selected features.

Figure 9

```
Private Sub g_Map_SelectionChanged()
  ' show the form if it's not visible
  If Not UserForm1.Visible Then
    UserForm1.Show vbModeless
  End If

  ' clear any previous results in the list
  UserForm1.ListBox1.Clear

  ' get the map's selection
  Dim activeView As IActiveView
  Set activeView = g_Map

  ' enumerate over the selected features
  Dim featureEnum As IEnumFeature
  Set featureEnum = activeView.Selection

  featureEnum.Reset  ' reset the cursor
  Dim feat As IFeature
  Set feat = featureEnum.Next
  Do While Not feat Is Nothing
    ' add the feature's name to the list
    Dim index As Long
    index = feat.Fields.FindField("Name")
    If index <> -1 Then
      UserForm1.ListBox1.AddItem feat.Value(index)
    End If

    Set feat = featureEnum.Next  ' get next feature
  Loop
End Sub
```

After showing the UserForm and clearing the ListBox, the script gets the Selection from the Map's IActiveView interface. The Selection's IEnumFeature interface is used to navigate through the collection of selected features, returning an IFeature interface. The code searches each feature's Fields collection for the Name field. If the Name field is present, the value is added to the ListBox.

Figure 10

```
Sub StartListening()
  Dim mxDoc As IMxDocument
  Set mxDoc = Application.Document
  Set g_Map = mxDoc.FocusMap
End Sub
```

The final step is to initialize the global variable, g_Map, as shown in Figure 10. Once this script has been run, the script in Figure 9 will be executed whenever the selection changes and the selections will be updated in the ListBox.

This is just an introduction to the many components that make up the ArcObjects object model. To get the latest developer information for ArcObjects and all ESRI developer products visit The Developer Connection at **www.esri.com/developer.** AU