

Library Reference

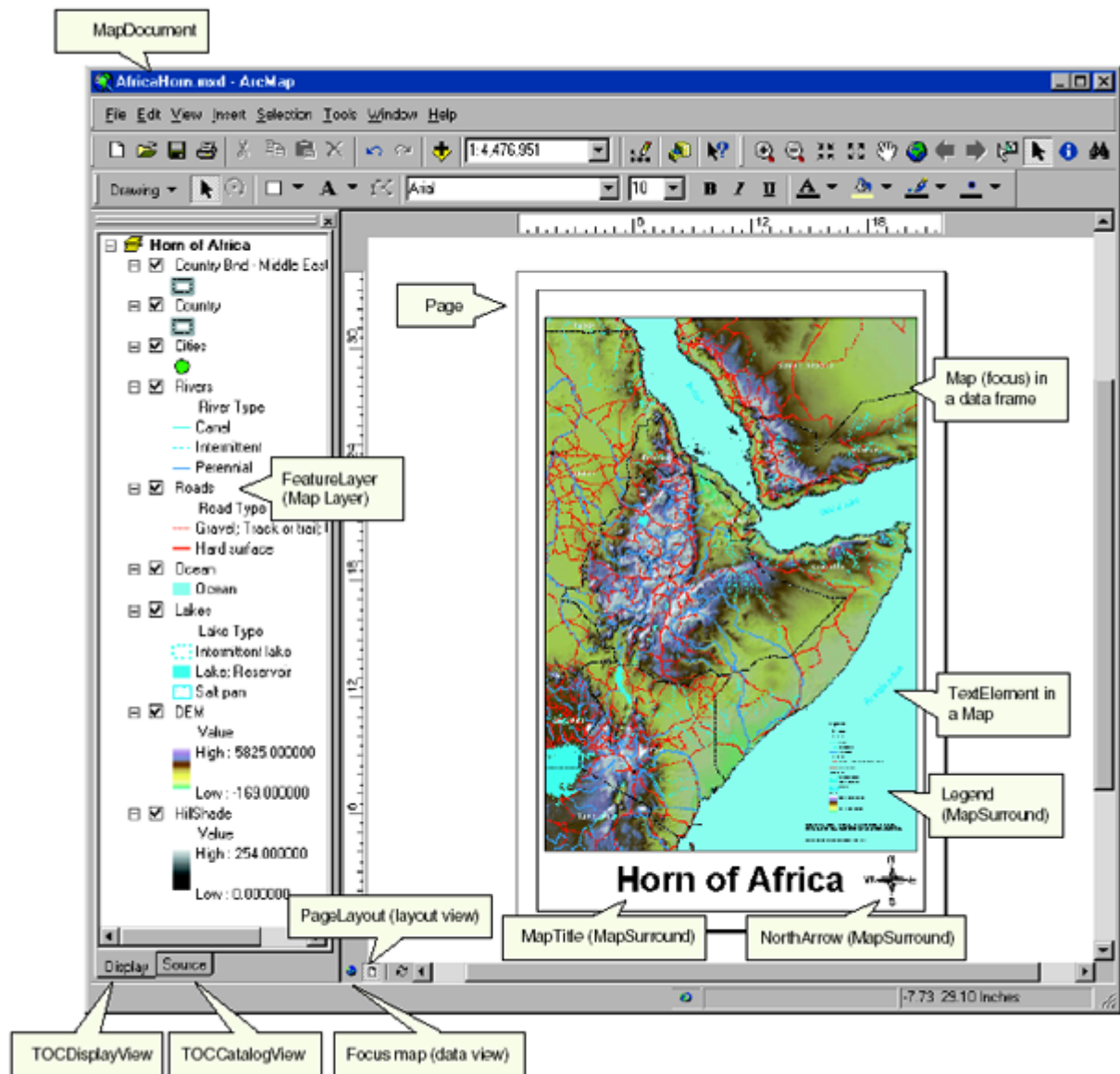
Carto Library Overview

Supported with: [ArcGIS Engine](#), [ArcGIS Desktop](#), [ArcGIS Server](#)

Library dependencies: [System](#), [SystemUI](#), [Geometry](#), [Display](#), [Server](#), [Output](#), [GeoDatabase](#), [GISClient](#), [DataSourcesFile](#), [DataSourcesGDB](#), [DataSourcesOleDB](#), [DataSourcesRaster](#), [GeoDatabaseDistributed](#)

Additional library information: [Contents](#), [Object Model Diagram](#)

The Carto library supports the creation and display of maps. ArcMap shows maps in two different views: the data view and the layout view. The data view will show the data from one given map whereas the page layout may display many maps and associated marginalia.



Key parts of the ArcMap user interface in page layout view.

You can use the *Map* object to access the map in the data view and the data it contains, and the *PageLayout* object to manage the map in the layout view and the map surrounds. But although developers can directly make use of the *Map* or *PageLayout* objects in their applications, it is more common for developers to use a higher level object such as the *MapControl*, *PageLayoutControl* or *ArcGIS* desktop application. These higher level objects simplify some tasks, although they always provide access to the lower level *Map* and *PageLayout* objects allowing the developer fine control of the objects.

The *PageLayout* object is a container for hosting one or more maps and their associated marginalia; North arrows, legends, scale bars, etc. The *Map* object is a container for layers. The *Map* object has properties that operate on all layers within the map; spatial reference, map scale, etc., along with methods that manipulate the map's layers.

There are many different types of layers that can be added to a map. Different data sources often have an associated layer responsible for displaying the data on the map; vector features are handled by the *FeatureLayer* object, raster data by the *RasterLayer*, TIN data by the *TinLayer*, etc. Layers can, if required, handle all the drawing operations for their associated data but it is more common for layers to have an associated *Renderer* object. The properties of the *Renderer* object control how the data is displayed in the map. Renderers commonly use symbols from the [Display](#) library for the actual drawing: the renderer simply matches a particular symbol with the properties of the entity that is to be drawn.

The *Map* and *PageLayout* objects are not the only objects in the Carto library that expose the behavior of map and page drawing. The *MxdServer* and *MapServer* objects both support the rendering of maps and pages, but instead of rendering to a window, these objects output directly to a file.

Using the *MapDocument* object developers can persist the state of the *Map* and *PageLayout* within a map document (.MXD), which can be used in ArcMap or one of the ArcGIS controls.

A *Map*, along with a *PageLayout*, can contain elements. An element has a geometry to define its location on the map or page, along with behaviors that control the display of the element. There are elements for basic shapes, text labels, complex marginalia, etc. The Carto library also contains support for map annotation and dynamic labeling.

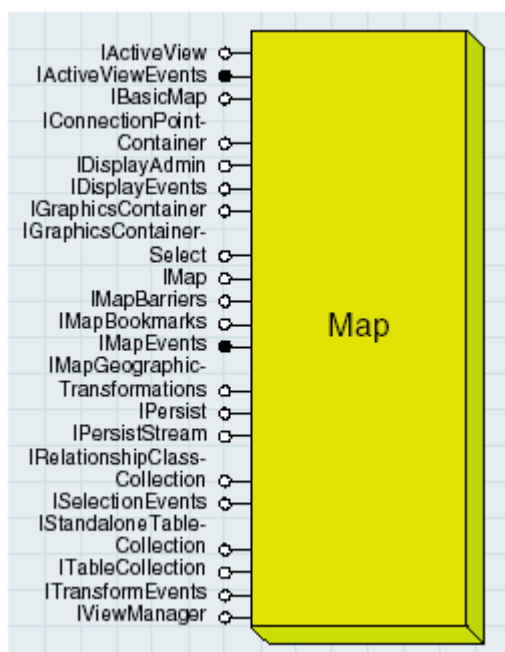
The Carto library is commonly extended in a number of areas. Custom renderers, layers, etc. are common. A custom layer is often the easiest method of adding custom data support to a mapping application.

The objects of the Carto library are grouped into a number of library subsystems. These library subsystems are:

- [Map and page layout](#)
- [Map elements](#)
- [Map surrounds](#)
- [Map grids](#)
- [Renderers](#)
- [Labeling](#)
- [Annotation](#)
- [Dimensions](#)
- [Layers](#)
- [MapServer](#)
- [ArcIMS layers, symbols, and renderers](#)
- [GPS support](#)

Map and page layout

The *Map* coclass



The *Map* object is a container for map data. It manages layers of features and graphic data. The *Map*

object is a primary point for customization tasks because it not only manages layers of data, but it is also a view and has to manage the drawing of all its data. Typical tasks with the *Map* object include adding a new layer, panning the display, changing the view extent (zooming functions), changing the spatial reference, and getting the currently selected features and elements.

The *Map* object is cocreatable so that new *Map* objects can be created and added to the document. Instantiating a new *Map* object automatically creates the following related objects on which it relies: a *ScreenDisplay* object, which every view uses to manage the drawing window, and a new *CompositeGraphicsLayer* as discussed [below](#).

The *IMap* interface

IMap : IUnknown	Provides access to members that control the map.
<ul style="list-style-type: none"> ■ ActiveGraphicsLayer: ILayer ■ AnnotationEngine: IAnnotateMap ■ AreaOfInterest: IEnvelope ■ Barriers (pExtent: IEnvelope) : IBarrierCollection ■ BasicGraphicsLayer: IGraphicsLayer ■ ClipBorder: IBorder ■ ClipGeometry: IGeometry ■ Description: String ■ DistanceUnits: esriUnits ■ Expanded: Boolean ■ FeatureSelection: ISelection ■ IsFramed: Boolean ■ Layer (in Index: Long) : ILayer ■ LayerCount: Long ■ Layers (UID: IUID, recursive: Boolean) : IEnumLayer ■ MapScale: Double ■ MapSurround (in Index: Long) : IMapSurround ■ MapSurroundCount: Long ■ MapUnits: esriUnits ■ Name: String ■ ReferenceScale: Double ■ SelectionCount: Long ■ SpatialReference: ISpatialReference ■ SpatialReferenceLocked: Boolean ■ UseSymbolLevels: Boolean 	<p>The active graphics layer. If no graphic layers exist a basic memory graphics layer will be created.</p> <p>The annotation (label) engine the map will use.</p> <p>Area of interest for the map.</p> <p>The list of barriers and their weight for labeling.</p> <p>The basic graphics layer.</p> <p>An optional border drawn around ClipGeometry.</p> <p>A shape that layers in the map are clipped to.</p> <p>Description of the map.</p> <p>The distance units for the map.</p> <p>Indicates if the Map is expanded.</p> <p>The feature selection for the map.</p> <p>Indicates if map is drawn in a frame rather than on the whole window.</p> <p>The layer at the given index.</p> <p>Number of layers in the map.</p> <p>The layers in the map of the type specified in the uid. If recursive is true it will return layers in group layers.</p> <p>The scale of the map as a representative fraction.</p> <p>The map surround at the given index.</p> <p>Number of map surrounds associated with the map.</p> <p>The units for the map.</p> <p>Name of the map.</p> <p>The reference scale of the map as a representative fraction.</p> <p>Number of selected features.</p> <p>The spatial reference of the map.</p> <p>Prevents the spatial reference from being changed.</p> <p>Indicates if the Map draws using symbol levels.</p>
<ul style="list-style-type: none"> ← AddLayer (in Layer: ILayer) ← AddLayers (in Layers: IEnumLayer, in autoArrange: Boolean) ← AddMapSurround (in MapSurround: IMapSurround) ← ClearLayers ← ClearMapSurrounds ← ClearSelection ← ComputeDistance (in p1: IPoint, in p2: IPoint) : Double ← CreateMapSurround (in CLSID: IUID, in optionalStyle: IMapSurround) : IMapSurround ← DelayDrawing (in delay: Boolean) ← DelayEvents (in delay: Boolean) ← DeleteLayer (in Layer: ILayer) ← DeleteMapSurround (in MapSurround: IMapSurround) ← GetPageSize (out widthInches: Double, out heightInches: Double) ← MoveLayer (in Layer: ILayer, in toIndex: Long) ← RecalcFullExtent ← SelectByShape (in Shape: IGeometry, in env: ISelectionEnvironment, in justOne: Boolean) ← SelectFeature (in Layer: ILayer, in Feature: IFeature) ← SetPageSize (in widthInches: Double, in heightInches: Double) 	<p>Adds a layer to the map.</p> <p>Adds multiple layers to the map, arranging them nicely if specified.</p> <p>Adds a map surround to the map.</p> <p>Removes all layers from the map.</p> <p>Removes all map surrounds from the map.</p> <p>Clears the map selection.</p> <p>Computes the distance between two points on the map and returns the result.</p> <p>Create and initialize a map surround. An optional style from the style gallery may be specified.</p> <p>Suspends drawing.</p> <p>Used to batch operations together to minimize notifications.</p> <p>Deletes a layer from the map.</p> <p>Deletes a map surround from the map.</p> <p>Gets the page size for the map.</p> <p>Moves a layer to another position.</p> <p>Forces the full extent to be recalculated.</p> <p>Selects features in the map given a shape and a selection environment (optional).</p> <p>Selects a feature.</p> <p>Sets the page size for the map (optional).</p>

The *IMap* interface is a starting point for many of the tasks you can do with a map. For example, you can use *IMap* to add, delete, and access map layers containing data from various sources, including feature layers and graphics layers; associate map surround objects (legends, scale bars, and so on) with the map; access the various properties of a map, including the area of interest, the current map units, and the spatial reference; select features and access the *Map* object's current selection.

The focus map

Every map document contains at least one *Map* object. Only one *Map* can have focus at a time, and this

Map is called the focus map. *IMxDocument* provides access to all of the *Map* objects loaded in the document; *IMxDocument::FocusMap* returns a reference to the *Map* currently with focus, and *IMxDocument::Maps* returns the entire collection of *Map* objects. A map document can contain any number of *Map* objects —the focus Map always represents the data view.

[Visual Basic 6.0]

```
Dim pMxDoc As IMxDocument
Set pMxDoc = Application.Document
Dim pMap As IMap
set pMap = pMxDoc.FocusMap
```

Accessing the map's layers

The *Map* object manages a collection of layer objects. Types of layer objects include *FeatureLayer*, *FDOGraphicsLayer*, and *GroupLayer*.

Each layer has a spatial reference. A spatial reference defines a precision and a coordinate system. The map coordinate system is automatically set to the coordinate system of the first layer loaded in the map and the precision is calculated based on the union of all the layers' extents. Refer to the [ISpatialReference](#) documentation for more information.

The following example shows how to add a layer based on a shapefile to the map.

[Visual Basic 6.0]

```
Public Sub AddShapeFile()
    Dim pWorkspaceFactory As IWorkspaceFactory
    Dim pFeatureWorkspace As IFeatureWorkspace
    Dim pFeatureLayer As IFeatureLayer
    Dim pMxDocument As IMxDocument
    Dim pMap As IMap

    ' Create a new ShapefileWorkspaceFactory object and
    ' open a shapefile folder - the path works with standard 9.0 installation
    Set pWorkspaceFactory = New ShapefileWorkspaceFactory
    Set pFeatureWorkspace = pWorkspaceFactory.OpenFromFile _
        ("C:\Program Files\ArcGIS\Bin\TemplateData\USA", 0)
    'Create a new FeatureLayer and assign a shapefile to it
    Set pFeatureLayer = New FeatureLayer
    Set pFeatureLayer.FeatureClass = _
        pFeatureWorkspace.OpenFeatureClass("States")
    pFeatureLayer.Name = pFeatureLayer.FeatureClass.AliasName
    'Add the FeatureLayer to the focus map
    Set pMxDocument = Application.Document
    Set pMap = pMxDocument.FocusMap
    pMap.AddLayer pFeatureLayer
End Sub
```

Selecting in the map

From the *Map* object you can find out about the selected features in the map:

[Visual Basic 6.0]

```
Public Sub GetSelectedFeature()
    Dim pMxDoc As IMxDocument
    Dim pMap As IMap
    Dim pEnumFeature As IEnumFeature
    Dim pFeature As IFeature

    Set pMxDoc = Application.Document
    Set pMap = pMxDoc.FocusMap
    Set pEnumFeature = pMap.FeatureSelection
    pEnumFeature.Reset
    Set pFeature = pEnumFeature.Next

    While Not pFeature Is Nothing
        Debug.Print pFeature.OID
        Set pFeature = pEnumFeature.Next
    Wend

End Sub
```

The *Map* object also contains methods to select features. The next code excerpt shows how to select features by shape.

[Visual Basic 6.0]

To use this sample, paste the code into VBA. Use the Commands tab of the Customize dialog to create a new *UIToolControl*. If you use the default name *UIToolControl1*, the code will be associated with the *MouseDown* event of the tool. Add the tool to any toolbar, select the tool and drag out an envelope. (You

will need to completely close VBA so that mouse events fire).

```
Private Sub UIToolControl1_MouseDown(ByVal button As Long, _
    ByVal shift As Long, ByVal x As Long, ByVal y As Long)

    Dim pMxApp As IMxApplication
    Dim pMxDoc As IMxDocument
    Dim pMap As IMap
    Dim pActiveView As IActiveView
    Dim pRubberEnv As IRubberBand
    Dim pEnvelope As IEnvelope
    Set pMxApp = Application 'QI
    Set pMxDoc = Application.Document
    Set pMap = pMxDoc.FocusMap
    Set pActiveView = pMap 'QI
    Set pRubberEnv = New RubberEnvelope

    'Flag the area of the old selection to invalidate
    pActiveView.PartialRefresh esriViewGeoSelection, Nothing, Nothing
    'Use TrackNew to prompt user to drag out a square on the display
    Set pEnvelope = pRubberEnv.TrackNew(pActiveView.ScreenDisplay, Nothing)
    'Perform the selection
    pMap.SelectByShape pEnvelope, pMxApp.SelectionEnvironment, False
    'Flag the area of the new selection to invalidate
    pActiveView.PartialRefresh esriViewGeoSelection, Nothing, Nothing

End Sub
```

Drawing on the Map's graphics layers

Map manages a *CompositeGraphicsLayer* object, which contains a collection of graphics layers.

The basic graphics layer is the default graphics layer of the *Map* where all graphics are drawn by default. *Map* provides direct access to this layer with the property *IMap::BasicGraphicsLayer*.

You can also access the *Map* object through the *IGraphicsContainer* interface to access its active graphics layer. This always returns a reference to *Map's* active graphics layer.

The *Map's* basic graphics layer is both a graphics layer on which to draw and the composite graphics layer which contains all the map's graphic layers. *Map's* basic graphics layer cannot be deleted from the *CompositeGraphicsLayer* object. Get a reference to the map's basic graphics layer through the *IGraphicsContainer* interface to manage the layer it contains. This way, graphics layers can be added to or deleted from the map.

The layer collection returned from the *IMap::Layers* property does not include the graphics layers managed by *Map's CompositeGraphicsLayer*. To access them, you can use the *IMap::ActiveGraphicsLayer* property. This property will return a reference to the graphics layer which is the current drawing target. This can either be the basic graphics layer, a layer in the *Map's CompositeGraphicsLayer*, or a feature layer such as an *FDOGraphicsLayer*.

This example adds a new graphics layer to the map. If you run this program and then have a look on the Annotation groups tab of the dataframe properties, you will see a new group i.e. a new graphics layer in the list. To draw in this new layer you must make it the active graphics layer. It comes as no surprise that this is done by setting the *ActiveGraphicsLayer* property of the map. You can tell which layer is the active graphics layer by going to the active annotation target sub-menu in the Drawing menu in ArcMap.

[Visual Basic 6.0]

```
Sub test()

    Dim pMxDoc As IMxDocument
    Set pMxDoc = ThisDocument
    Dim pMap As IMap
    Set pMap = pMxDoc.FocusMap
    Dim pGraphicsLayer As IGraphicsLayer
    Dim pCompositeGraphicsLayer As ICompositeGraphicsLayer
    Set pCompositeGraphicsLayer = pMap.BasicGraphicsLayer
    Set pGraphicsLayer = pCompositeGraphicsLayer.AddLayer("New Graphics Layer", Nothing)
    Set pMap.ActiveGraphicsLayer = pGraphicsLayer

End Sub
```

In the next example we *QueryInterface* from *IMap* to *IGraphicsContainer* to add a text element to the active graphics layer. For more on elements see the [Map Elements](#) section below.

[Visual Basic 6.0]

```
Dim pMxDoc As IMxDocument
Set pMxDoc = ThisDocument
Dim pMap As IMap
Set pMap = pMxDoc.FocusMap
```



```

Dim pGraphicsContainer As IGraphicsContainer
Set pGraphicsContainer = pMap
Dim pElement As IElement
Dim pTextElement As ITextElement
Set pElement = New TextElement
Set pTextElement = pElement
Dim pPoint As IPoint
Set pPoint = New Point
pPoint.x = 0
pPoint.y = 0
pElement.Geometry = pPoint
pTextElement.Text = "Hello World"
pGraphicsContainer.AddElement pElement, 0

```

IGraphicsContainer : IUnknown	Provides access to members that control the Graphics Container.
← AddElement (in Element: IElement, in zorder: Long)	Add a new graphic element to the layer.
← AddElements (in Elements: IElementCollection, in zorder: Long)	Add new graphic elements to the layer.
← BringForward (in Elements: IEnumElement)	Move the specified elements one step closer to the top of the stack of elements.
← BringToFront (in Elements: IEnumElement)	Make the specified elements draw in front of all other elements.
← DeleteAllElements	Delete all the elements.
← DeleteElement (in Element: IElement)	Delete the given element.
← FindFrame (in frameObject: Variant) : IFrameElement	Find the frame that contains the specified object.
← GetElementOrder (in Elements: IEnumElement) : Variant	Private order object. Used to undo ordering operations.
← LocateElements (in Point: IPoint, in Tolerance: Double) : IEnumElement	Returns the elements at the given coordinate.
← LocateElementsByEnvelope (in Envelope: IEnvelope) : IEnumElement	Returns the elements that intersect with the given envelope.
← MoveElementFromGroup (in Group: IGroupElement, in Element: IElement, in zorder: Long)	Move the element from the group to the container.
← MoveElementToGroup (in Element: IElement, in Group: IGroupElement)	Move the element from the container to the group.
← Next : IElement	Returns the next graphic in the container.
← PutElementOrder (in order: Variant)	Private order object. Used to undo ordering operations.
← Reset	Reset internal cursor so that Next returns the first element.
← SendBackward (in Elements: IEnumElement)	Move the specified elements one step closer to the bottom of the stack of elements.
← SendToBack (in Elements: IEnumElement)	Make the specified elements draw behind all other elements.
← UpdateElement (in Element: IElement)	The graphic element's properties have changed.

The basic graphics layer is a special layer that cannot be deleted and is not reported in the *CompositeGraphicsLayer's* layer count. Further, this layer's element count reports the total number of elements in all the *CompositeGraphicsLayer's* layers. If you delete all elements in *Map's* basic graphics layer, you delete all elements in all target layers (annotation groups) in the *CompositeGraphicsLayer*. In the case where the *Map's CompositeGraphicsLayer* does have multiple layers, use *IMap::ActiveGraphicsLayer* to set or get a reference to the active layer.

The active graphics layer does not always reference a layer in the *Map's CompositeGraphicsLayer*; this is the case when a database layer containing elements is set as the active graphics layer. A feature-linked annotation layer (*FDOGraphicsLayer*) is a good example of this.

The *Map's IGraphicsContainer* always returns a reference to the *Map's* active graphics layer. Again, this can either be the basic graphics layer, a layer in the *Map's CompositeGraphicsLayer*, or a feature layer such as an *FDOGraphicsLayer*.

See also the description of the *IGraphicsContainerSelect* interface under the Page Layout topic [below](#).

Map frames and surrounds

In ArcMap, *Map* objects are always contained by *MapFrame* objects. The *PageLayout* object actually manages all the *MapFrame* objects and each *MapFrame* manages a *Map*. Note that for convenience, the *MxDocument* object passes a reference to the focus map and the *Map's* collection. In reality, however, the *PageLayout* object manages these.

MapSurround objects are elements that are related to a *Map*. Types of map surrounds include legends, North arrows, and scale bars. The *Map* object exposes several properties and methods for accessing the

map surrounds associated with it. All map surrounds are actually contained by a *MapSurroundFrame* which, like a *MapFrame*, is ultimately managed by the *PageLayout* object.

See the page layout section [below](#) for details on the map frame and surrounds.

IBasicMap

IBasicMap : IUnknown	Provides access to members that control the basic map.
<ul style="list-style-type: none"> ▣ ActiveGraphicsLayer: ILayer ▣ AreaOfInterest: IEnvelope ▣ BasicGraphicsLayer: IGraphicsLayer ▣ Description: String ▣ FeatureSelection: ISelection ▣ Layer (in Index: Long) : ILayer ▣ LayerCount: Long ▣ Layers (UID: IUID, recursive: Boolean) : IEnumLayer ▣ Name: String ▣ SelectionCount: Long ▣ SpatialReference: ISpatialReference 	<p>The active graphics layer. If no graphic layers exist a basic memory graphics layer will be created.</p> <p>Area of interest for the map.</p> <p>The basic graphics layer.</p> <p>Description of the map.</p> <p>The map's feature selection.</p> <p>The layer at the given index.</p> <p>Number of layers in the map.</p> <p>The layers in the map of the type specified in the uid. If recursive is true it will return layers in group layers.</p> <p>Name of the map.</p> <p>Number of selected features in the map.</p> <p>The spatial reference of the map.</p>
<ul style="list-style-type: none"> ← AddLayer (in pLayer: ILayer) ← AddLayers (in pLayers: IEnumLayer, in autoArrange: Boolean) ← ClearLayers ← ClearSelection ← DeleteLayer (in pLayer: ILayer) ← SelectByShape (in Shape: IGeometry, in env: ISelectionEnvironment, in justOne: Boolean) 	<p>Adds a layer to the map.</p> <p>Adds multiple layers to the map, arranging them nicely if specified.</p> <p>Removes all layers from the map.</p> <p>Clears the map selection.</p> <p>Deletes a layer from the map.</p> <p>Selects features in the map given a shape and a selection environment (optional).</p>

IBasicMap is a subset of *IMap* that provides support for ArcScene and ArcGlobe. The *Map* (2D), *Scene* (3D), and *Globe* (3D) coclasses implement this interface. Components used by ArcMap, ArcScene, and ArcGlobe, (such as the *IdentifyDialog*) utilize *IBasicMap* rather than *IMap*.

The active view

IActiveView : IUnknown	Provides access to members that control the active view - the main application window.
<ul style="list-style-type: none"> ExportFrame: tagRECT Extent: IEnvelope ExtentStack: IExtentStack FocusMap: IMap FullExtent: IEnvelope GraphicsContainer: IGraphicsContainer IsMapActivated: Boolean ScreenCacheID (in phase: tagesriViewDrawPhase, in data: IUnknown Pointer) : Integer ScreenDisplay: IScreenDisplay Selection: ISelection ShowRulers: Boolean ShowScrollBars: Boolean ShowSelection: Boolean TipText (in X: Double, in Y: Double) : String 	<p>The device rectangle to export.</p> <p>The visible extent rectangle.</p> <p>The extent stack.</p> <p>The map that tools and controls act on.</p> <p>The full extent rectangle.</p> <p>The active graphics container.</p> <p>Indicates if the focus map is activated.</p> <p>The screen cache ID that is used to draw the specified phase.</p> <p>The screen display used by the view.</p> <p>The selection.</p> <p>Indicates if rulers are visible.</p> <p>Indicates if scrollbars are visible.</p> <p>Indicates if selection is visible.</p> <p>The tip text to display at the given location.</p>
<ul style="list-style-type: none"> Activate (hWnd: Long) Clear ContentsChanged Deactivate Draw (in hDC: Long, in trackCancel: ITrackCancel) GetContextMenu (in X: Double, in Y: Double, out clsidMenu: IUID) HitTestMap (in Location: IPoint) : IMap IsActive: Boolean OnMessage (in msg: Unsigned Long, in wParam: Unsigned Machine Int, in lParam: Long) Output (in hDC: Long, in dpi: Long, in PixelBounds: tagRECT, in VisibleBounds: IEnvelope, in trackCancel: ITrackCancel) PartialRefresh (in phase: tagesriViewDrawPhase, in data: IUnknown Pointer, in Envelope: IEnvelope) PrinterChanged (in Printer: IPrinter) Refresh 	<p>Gives this view control of the specified window.</p> <p>Empties the view contents.</p> <p>Called by clients when view objects are modified.</p> <p>Another view takes over the associated window.</p> <p>Draws the view to the specified device context. TrackCancel is optional.</p> <p>Called when a context menu should be displayed at the given xy location. Return menu that should be displayed.</p> <p>Returns any maps present in the view at the given location. Return value may be zero if there are no maps or the coordinate is not over a map.</p> <p>Indicates if view is active or not.</p> <p>Call from your application's message loop to enable automatic resizing and keyboard accelerators.</p> <p>Renders the view to the specified DC.</p> <p>Draws the specified view phase. Use an envelope of zero to draw the entire phase.</p> <p>Called by application when printer changes.</p> <p>Causes the entire view to draw.</p>

The *IActiveView* interface controls the main application window, including all drawing operations. Use this interface to change the extent of the view, access the associated *ScreenDisplay* object, show or hide rulers and scroll bars, and refresh the view. Refreshing the view is discussed in detail in the [Display Library Overview](#) and in the help for the [Refresh](#) and [PartialRefresh](#) methods of *IActiveView*.

This VBA script lets the user zoom in on the current active view:

[Visual Basic 6.0]

```
Public Sub ZoomInCenter()
    Dim pMxDocument As IMxDocument
    Dim pActiveView As IActiveView
    Set pMxDocument = Application.Document

    ' try these different options
    Set pActiveView = pMxDocument.ActiveView
    ' Set pActiveView = pMxDocument.PageLayout
    ' Set pActiveView = pMxDocument.FocusMap
    ' to see how this program works in data view or layout view

    Dim pEnvelope As IEnvelope
    Dim pCenterPoint As IPoint
    ' get a copy of the envelope corresponding to the view extent
    Set pEnvelope = pActiveView.Extent
    Set pCenterPoint = New Point
    pCenterPoint.x = ((pEnvelope.XMax - pEnvelope.XMin) / 2) + pEnvelope.XMin
    pCenterPoint.y = ((pEnvelope.YMax - pEnvelope.YMin) / 2) + pEnvelope.YMin
    pEnvelope.Width = pEnvelope.Width / 2 ' resize envelope width
    pEnvelope.Height = pEnvelope.Height / 2 ' resize envelope width
```

```

pEnvelope.CenterAt pCenterPoint ' move envelope to center point
pActiveView.Extent = pEnvelope ' put envelope in view extent
pActiveView.Refresh
End Sub

```

Active view events

IActiveViewEvents : IUnknown	Provides access to events that occur when the state of the active view changes.
<ul style="list-style-type: none"> ← AfterDraw (in Display: IDisplay, in phase: tagesriViewDrawPhase) ← AfterItemDraw (in Index: Integer, in Display: IDisplay, phase: tagesriDrawPhase) ← ContentsChanged ← ContentsCleared ← FocusMapChanged ← ItemAdded (in Item: Variant) ← ItemDeleted (in Item: Variant) ← ItemReordered (in Item: Variant, in toIndex: Long) ← SelectionChanged ← SpatialReferenceChanged ← ViewRefreshed (in View: IActiveView, in phase: tagesriViewDrawPhase, in data: Variant, in Envelope: IEnvelope) 	<p><i>Fired after the specified phase is drawn.</i></p> <p><i>Fired after an individual view item is drawn. Example: view items include layers in a map or elements in a page layout.</i></p> <p><i>Fired when the contents of the view changes.</i></p> <p><i>Fired when the contents of the view is cleared.</i></p> <p><i>Fired when a new map is made active.</i></p> <p><i>Fired when an item is added to the view.</i></p> <p><i>Fired when an item is deleted from the view.</i></p> <p><i>Fired when a view item is reordered.</i></p> <p><i>Fired when the selection changes.</i></p> <p><i>Fired when the spatial reference is changed.</i></p> <p><i>Fired when view is refreshed before draw happens.</i></p>

The *IActiveViewEvents* interface is the default outbound interface on the *Map* object. It is exposed by the *Map* object so that clients may listen and respond to specific events related to the active view, such as *AfterDraw* and *SelectionChanged*.

Many coclasses implement this interface, and each of them fires events differently. The *Map* object's implementation of the *IActiveView* is different from the *PageLayout* object's implementation. For example, the *Map* object does not fire the *FocusMap Changed* event, whereas the *PageLayout* object does. Similarly, the *Map* object fires the *Item Deleted* event when a layer is removed from the *Map*, and the *PageLayout* object fires the same event when elements such as a map frame or graphic are deleted. The *AfterViewDraw* event will not fire unless *IViewManager::VerboseEvents* is set to *True*.

To use this example, paste the code into VBA then run the *SetUpEvents* procedure. The *SelectionChanged* will be fired and the message displayed each time the selection is modified.

[Visual Basic 6.0]

```

Private WithEvents MapActiveViewEvents As Map

Public Sub SetUpEvents()
    Dim pMxDoc As IMxDocument
    Set pMxDoc = Application.Document
    Set MapActiveViewEvents = pMxDoc.FocusMap
End Sub

Private Sub MapActiveViewEvents_SelectionChanged()
    MsgBox "Selection Changed"
End Sub

```

IViewManager

IViewManager : IUnknown	Provides access to members used to describe or define view behavior.
ConserveMemory : Boolean	Indicates whether to be conservative when allocating resources.
DelayBackgroundDraw : Boolean	Indicates if the background should draw immediately. Set to true to eliminate flashing during animation.
ElementSelection : ISelection	Object to use for element selection.
ExternalDrawing (in phase: <i>tagesriViewDrawPhase</i>): Boolean	Indicates if external clients are drawing in response to the specified phase.
OutputBandSize : Long	Size allocated for each band when banding output.
TopFilterIndex : Long	Phase index that supplements <i>TopFilterPhase</i> . Clients should set the item index here if they draw in response to <i>AfterDrawItem</i> and they use a display filter. <i>TopFilterPhase</i> must also be specified.
TopFilterPhase : <i>tagesriViewDrawPhase</i>	The highest phase in the drawing order that uses a display filter. Clients should set this when they draw in response to <i>AfterDraw</i> and they use a display filter.
UsesPageCoordinates : Boolean	Indicates whether view uses page coordinates.
VerboseEvents : Boolean	Expand or limit the number of events that are fired. The following events are not fired if <i>VerboseEvents</i> is false: <i>AfterDrawItem</i> .

IViewManager is a low-level interface to the properties defining the behavior of the active view.

One commonly used property managed by the *IViewManager* interface is *VerboseEvents*. When *VerboseEvents* is set to False, the default, *IActiveViewEvents::AfterItemDraw*, is not fired. To listen for this event, you must set *VerboseEvents* equal to True.

The sample below buffers each selected feature and draws the result on the display. The buffer polygons have a black outline and a slanted red line fill. Paste the code into VBA and run the *InitEvents* procedure.

[Visual Basic 6.0]

```
Private WithEvents ActiveViewEvents As Map
Private m_pMxDoc As IMxDocument
Private m_pBufferPolygon As IPolygon
Private m_pLastBufferedExtent As IEnvelope
Private m_pFillSymbol As ISimpleFillSymbol

Public Sub InitEvents()
    Dim pViewManager As IViewManager
    Dim pRgbColor As IRgbColor
    Set m_pMxDoc = Application.Document
    Set pViewManager = m_pMxDoc.FocusMap
    pViewManager.VerboseEvents = True
    Set ActiveViewEvents = m_pMxDoc.FocusMap
    Set m_pActiveView = m_pMxDoc.FocusMap
    'Create a fill symbol
    Set m_pFillSymbol = New SimpleFillSymbol
    Set pRgbColor = New RgbColor
    pRgbColor.Red = 255
    m_pFillSymbol.Style = esriSFSForwardDiagonal
    m_pFillSymbol.Color = pRgbColor
End Sub

Private Sub ActiveViewEvents_AfterItemDraw(ByVal Index As Integer, _
    ByVal display As IDisplay, ByVal phase As esriDrawPhase)
    'Only draw in the geography phase
    If Not phase = esriDPGeography Then Exit Sub
    'Draw the buffered polygon
    If m_pBufferPolygon Is Nothing Then Exit Sub
    With display
        .SetSymbol m_pFillSymbol
        .DrawPolygon m_pBufferPolygon
    End With
End Sub

Private Sub ActiveViewEvents_SelectionChanged()
    Dim pActiveView As IActiveView
    Dim pEnumFeature As IEnumFeature
    Dim pFeature As IFeature
    Dim pPolygon As IPolygon
    Dim pTopoOperator As ITopologicalOperator
    Dim pGeometryBag As IGeometryCollection
    Set pActiveView = m_pMxDoc.FocusMap
    Set pGeometryBag = New GeometryBag
    'Flag last buffered region for invalidation
    If Not m_pLastBufferedExtent Is Nothing Then
```


```

    pActiveView.PartialRefresh esriViewGeography, Nothing, _
    m_pLastBufferedExtent
End If
If m_pMxDoc.FocusMap.SelectionCount = 0 Then
'Nothing selected; don't draw anything; bail
    Set m_pBufferPolygon = Nothing
    Exit Sub
End If
'Buffer each selected feature
    Set pEnumFeature = m_pMxDoc.FocusMap.FeatureSelection
    pEnumFeature.Reset
    Set pFeature = pEnumFeature.Next
    Do While Not pFeature Is Nothing
        Set pTopoOperator = pFeature.Shape
        Set pPolygon = pTopoOperator.Buffer(0.1)
        pGeometryBag.AddGeometry pPolygon
    'Get next feature
        Set pFeature = pEnumFeature.Next
    Loop

'Union all the buffers into one polygon
    Set m_pBufferPolygon = New Polygon
    Set pTopoOperator = m_pBufferPolygon 'QI
    pTopoOperator.ConstructUnion pGeometryBag
    Set m_pLastBufferedExtent = m_pBufferPolygon.Envelope
'Flag new buffered region for invalidation
    pActiveView.PartialRefresh esriViewGeography, Nothing, m_pBufferPolygon.Envelope
End Sub

```





Barriers

IMapBarriers : IUnknown	Provides access to members that control map barriers.
 Barriers2 (pExtent: IEnvelope, in pTrackCancel: ITrackCancel) : IBarrierCollection	<i>The list of barriers and their weight for labeling.</i>

Barriers are used by labeling engines to signal that a label should not be placed in a particular region. Barriers currently include annotation, graphical elements, and symbols generated from renderers. For example, a feature layer using a pie chart renderer doesn't want labels to appear directly above the pie chart's symbols. In this case, pie chart symbols act as barriers informing the label engine that no labels should be placed on top of them.

The *IMapBarriers* interface returns a list of all the barriers and their weights from all the layers in the *Map*. Layers with barriers include those layers that implement *IBarrierProperties*—the *CompositeGraphicsLayer*, *CoverageAnnotationLayer*, and *FDOGraphicsLayer*. When creating a labeling engine, use this interface to conveniently access all the barriers from all the layers.

Spatial bookmarks

IMapBookmarks : IUnknown	Provides access to members that control the map bookmarks.
 Bookmarks : IEnumSpatialBookmark	<i>The bookmarks.</i>
 AddBookmark (in bookmark: ISpatialBookmark)	<i>Adds a bookmark to the collection.</i>
 RemoveAllBookmarks	<i>Removes all bookmarks.</i>
 RemoveBookmark (in bookmark: ISpatialBookmark)	<i>Removes a bookmark from the collection.</i>

All spatial bookmarks are managed by and are persisted in the *Map* object. Bookmarks save map extents along with a name identifying them and so make it easy to jump to a specific location on the map. In ArcMap, bookmarks are accessible under Bookmarks in the View menu.

Map's bookmarks are managed by the *IMapBookmarks* interface. Use *IMapBookmarks* to access existing bookmarks, add new ones, and delete old ones. Once you have a reference to a particular bookmark, you can make the *Map's* extent equal to that stored in the bookmark. There are two types of spatial bookmarks available in ArcMap: Area of Interest bookmarks and Feature bookmarks. Area of Interest bookmarks store information about a map extent, Feature bookmarks allow you to find back one particular feature on the map. See the help under [ISpatialBookmark](#), [IAOIBookmark](#) and [IFeatureBookmark](#) for more on these.

This sample shows one method for creating a new Area of Interest bookmark:

[Visual Basic 6.0]



```
Public Sub AddSpatialBookMark()
    Dim pMxDoc As IMxDocument
    Dim pMap As IMap
    Dim pActiveView As IActiveView
    Dim pAreaOfInterest As IAOIBookmark
    Dim pMapBookmarks As IMapBookmarks
    Set pMxDoc = Application.Document
    Set pMap = pMxDoc.FocusMap
    Set pActiveView = pMap
    'Create a new bookmark and set its location to the focus map's
    'current extent
    Set pAreaOfInterest = New AOIBookmark
    Set pAreaOfInterest.Location = pActiveView.Extent
    'Give the bookmark a name
    pAreaOfInterest.Name = "Aera of interest bookmark"
    'Add the bookmark to the map's bookmark collection.This will add
    'the bookmark to the Bookmarks menu accessible from the View menu
    Set pMapBookmarks = pMap
    pMapBookmarks.AddBookmark pAreaOfInterest
End Sub
```

This sample shows one way to find an existing spatial bookmark and zoom to its stored extent:

[Visual Basic 6.0]

```
Public Sub ZoomToBookmark()
    Dim pMxDoc As IMxDocument
    Set pMxDoc = Application.Document
    ' Access the Map object through its IMapBookmarks interface
    Dim pMapBookmarks As IMapBookmarks
    Set pMapBookmarks = pMxDoc.FocusMap
    ' get an enumeration of bookmarks
    Dim pEnumBookmarks As IEnumSpatialBookmark
    Set pEnumBookmarks = pMapBookmarks.Bookmarks
    pEnumBookmarks.Reset
    Dim pBookmark As ISpatialBookmark
    ' parse the enum
    Set pBookmark = pEnumBookmarks.Next
    Do While Not pBookmark Is Nothing
        If pBookmark.Name = "Aera of interest bookmark" Then
            pBookmark.ZoomTo pMxDoc.FocusMap ' zoom to bookmark
            pMxDoc.ActiveView.Refresh
            Exit Sub
        End If
        Set pBookmark = pEnumBookmarks.Next
    Loop
End Sub
```

Map events

IMapEvents : IUnknown	<i>Provides access to events that occur when the state of the map changes.</i>
 FeatureClassChanged (in oldClass: IFeatureClass, in newClass: IFeatureClass)	<i>Fired when the feature class changes.</i>
 VersionChanged (in oldVersion: IVersion, in newVersion: IVersion)	<i>Fired when the version changes.</i>

The *IMapEvents* interface is exposed off the *Map* object, enabling clients to listen and respond to two events occurring inside a map: *FeatureClassChanged* and *VersionChanged*. Both of these events are related to changing the version the map's layers are working with. For example, if someone changes the version an edit session is working with, the Editor has to know about all the new feature classes so that it can reset the snapping environment.

The *Map* object's default outbound interface is *IActiveViewEvents*. Because Visual Basic can only handle one outbound interface per object, the *MapEvents* object has been created to give Visual Basic users a method for responding to the events grouped under *IMapEvents*.

The example demonstrates listening to map events. The event is declared on the *MapEvents* object instead of the *Map* object. Paste into VBA and run the *InitEvents* procedure.

[Visual Basic 6.0]

```
Private WithEvents MapEvents As MapEvents

Public Sub InitEvents()
```



```






Dim pMxDoc As IMxDocument
Set pMxDoc = Application.Document
Set MapEvents = pMxDoc.FocusMap
End Sub

Private Sub MapEvents_FeatureClassChanged(ByVal oldClass As _
    IFeatureClass, ByVal newClass As IFeatureClass)
    MsgBox "Feature Class Changed"
End Sub

Private Sub MapEvents_VersionChanged(ByVal oldVersion As _
    IVersion, ByVal newVersion As IVersion)
    MsgBox "Version Changed"
End Sub

```

ITableCollection

ITableCollection : IUnknown	Provides access to members that control a table collection.
 Table (in Index: Long) : ITable  TableCount: Long	<i>The table at the given index.</i> <i>Number of tables.</i>
 AddTable (in Table: ITable)  RemoveAllTables  RemoveTable (in Table: ITable)	<i>Adds a table to the collection.</i> <i>Removes all tables from the collection.</i> <i>Removes a table from the collection.</i>

The *ITableCollection* interface is used to manage tables associated with a *Map*. Use this interface to add new tables to a map, remove old tables, or access a table already loaded.

The following VBA macro loads a table into the focus map.

[Visual Basic 6.0]

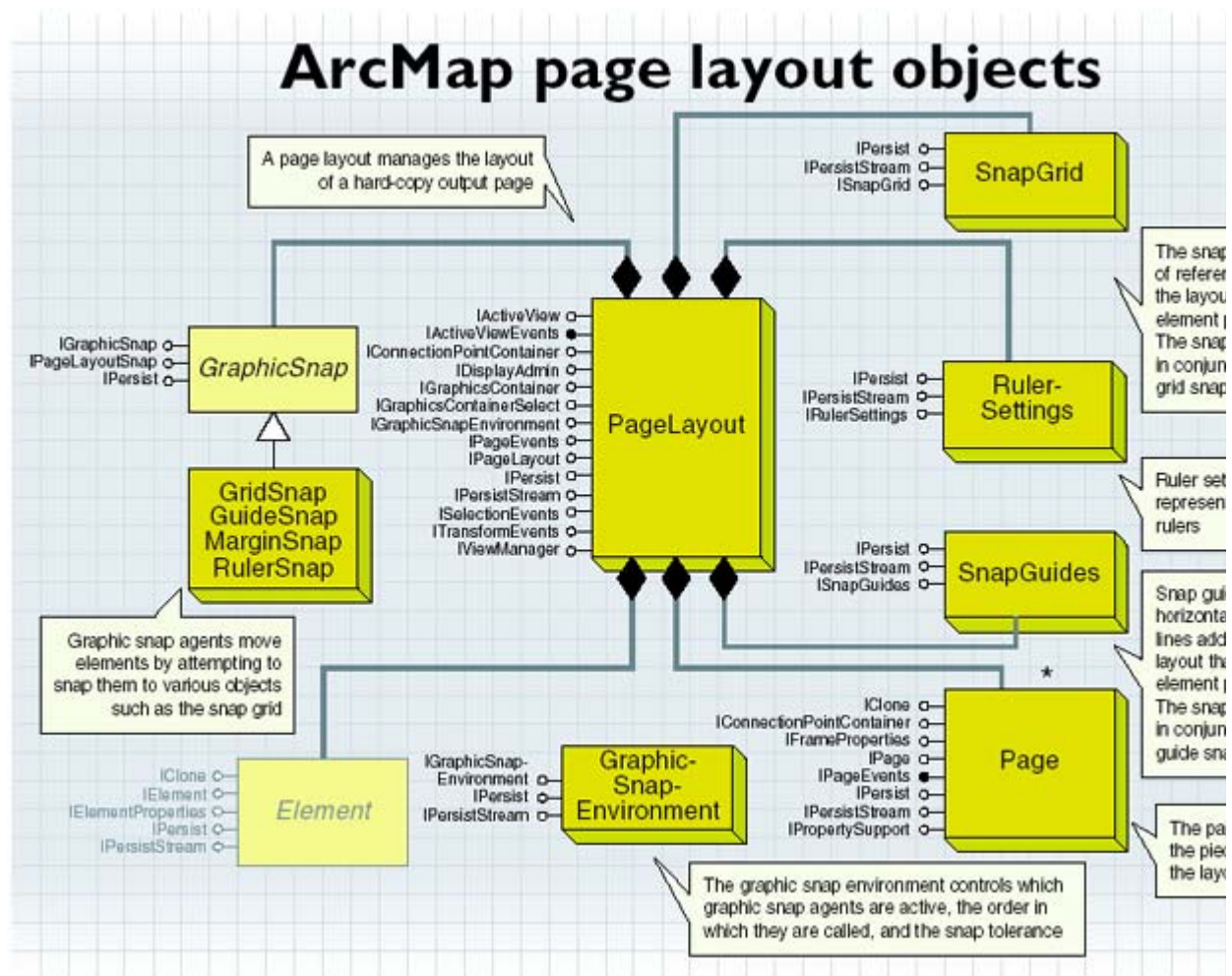
```

Public Sub AddTable()
    Dim pMxDoc As IMxDocument
    Dim pMap As IMap
    Dim pTable As ITable
    Dim pTableCollection As ITableCollection
    Set pMxDoc = Application.Document
    Set pMap = pMxDoc.FocusMap
    Set pTableCollection = pMap 'QI
    Set pTable = OpenTable("C:\Program Files\ArcGIS\Bin\TemplateData\USA", "states.dbf")
    If pTable Is Nothing Then Exit Sub
    pTableCollection.AddTable pTable
    pMxDoc.UpdateContents
End Sub

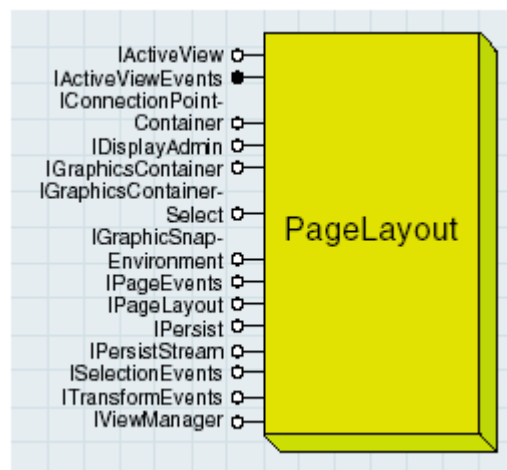
Private Function OpenTable(strWorkspace As String, strTableName As String) As ITable
    On Error GoTo ErrorHandler
    Dim pShpWorkspaceName As IWorkspaceName
    Dim pDatasetName As IDatasetName
    Dim pName As IName
    'Create the workspace name object
    Set pShpWorkspaceName = New WorkspaceName
    pShpWorkspaceName.PathName = strWorkspace
    pShpWorkspaceName.WorkspaceFactoryProgID = "esriCore.shapefileworkspacefactory.1"
    'Create the table name object
    Set pDatasetName = New TableName
    pDatasetName.Name = strTableName
    Set pDatasetName.WorkspaceName = pShpWorkspaceName
    'Open the table
    Set pName = pDatasetName
    Set OpenTable = pName.Open
    Exit Function 'exit to avoid error handler
ErrorHandler:
    Set OpenTable = Nothing
End Function

```

The page layout



The *PageLayout* coclass



The *PageLayout* object corresponds to the ArcMap layout view. A *PageLayout* object is automatically created by the document when you first start ArcMap. Access the ArcMap *PageLayout* object via *IMxDocument::PageLayout*.

The *PageLayout* object is very similar to the *Map* object. Both are views, meaning they take control of the main application window; both are also graphics containers, meaning they can store graphical elements. If there is no map activated in layout view (*IMxDocument::ActivatedView*), all new graphic elements are added to the *PageLayout*. If a *Map* is activated, graphic elements are added to the focus map (*IMxDocument::FocusMap*).

Although both the *PageLayout* and *Map* objects are graphics containers, the type of graphics they store is different. The *PageLayout* can additionally store frame elements such as a *MapFrame*, and both can store graphic elements, such as a *TextElement*. Although the map document (*MxDocument*) can pass a reference to the focus map and the entire collection of maps in the document, the *PageLayout* object really manages

all *Map* objects via *MapFrame* objects. In ArcMap, all *Maps* must be contained by a *MapFrame* element, which is directly managed by the *PageLayout*. It is only for convenience that map documents are able to pass a reference to *Maps*.

In order to present itself as a hardcopy output page, the *PageLayout* automatically creates these objects: *SnapGuides*, *SnapGrid*, *RulerSettings*, and *Page*.

The *IPageLayout* interface

IPageLayout : IUnknown	Provides access to members that control the Page Layout.
<ul style="list-style-type: none"> ■ AlignToMargins: Boolean ■ HorizontalSnapGuides: ISnapGuides ■ Page: IPage ■ RulerSettings: IRulerSettings ■ SnapGrid: ISnapGrid ■ VerticalSnapGuides: ISnapGuides ■ ZoomPercent: Double 	<p><i>Indicates if graphics will be aligned to the margins or to each other.</i></p> <p><i>The horizontal snapping guides.</i></p> <p><i>The page.</i></p> <p><i>The ruler settings.</i></p> <p><i>The snapping grid.</i></p> <p><i>The vertical snapping guides.</i></p> <p><i>The current zoom percent. 100 means 1:1. 200 means twice normal size, etc.</i></p>
<ul style="list-style-type: none"> ← FocusNextMapFrame ← FocusPreviousMapFrame ← ReplaceMaps (in Maps: IMaps) ← ZoomToPercent (in percent: Long) ← ZoomToWhole ← ZoomToWidth 	<p><i>Focus the next map.</i></p> <p><i>Focus the previous map.</i></p> <p><i>Replace the maps in the data frames with the specified maps. If there are more maps than frames, new frames are created. If there are fewer frames than maps, extra frames are cleared.</i></p> <p><i>Magnify the page by a certain percentage. 100 means actual size. 200 means twice normal size, etc.</i></p> <p><i>Fit the whole page in the window.</i></p> <p><i>Fit the width of the page to the screen.</i></p>

The *IPageLayout* interface is the primary interface implemented by the *PageLayout* object. Use this interface to access the *RulerSettings*, the *SnapGrid*, the *SnapGuides*, and the *Page* objects. *IPageLayout* also has methods for zooming the view and changing the focus map. This code demonstrates zooming.

[Visual Basic 6.0]

```
Public Sub ZoomToPercent()
    Dim pPageLayout As IPageLayout
    Dim pMxDoc As IMxDocument
    Set pMxDoc = Application.Document
    Set pPageLayout = pMxDoc.PageLayout
    'Ensure the application is in layout view
    If Not pMxDoc.ActiveView Is pMxDoc.PageLayout Then _
        Set pMxDoc.ActiveView = pMxDoc.PageLayout
    pPageLayout.ZoomToPercent 50 'Zoom the view to 50%
End Sub
```

IGraphicsContainer

IGraphicsContainer : IUnknown	Provides access to members that control the Graphics Container.
← AddElement (in Element: IElement, in zorder: Long)	Add a new graphic element to the layer.
← AddElements (in Elements: IElementCollection, in zorder: Long)	Add new graphic elements to the layer.
← BringForward (in Elements: IEnumElement)	Move the specified elements one step closer to the top of the stack of elements.
← BringToFront (in Elements: IEnumElement)	Make the specified elements draw in front of all other elements.
← DeleteAllElements	Delete all the elements.
← DeleteElement (in Element: IElement)	Delete the given element.
← FindFrame (in frameObject: Variant) : IFrameElement	Find the frame that contains the specified object.
← GetElementOrder (in Elements: IEnumElement) : Variant	Private order object. Used to undo ordering operations.
← LocateElements (in Point: IPoint, in Tolerance: Double) : IEnumElement	Returns the elements at the given coordinate.
← LocateElementsByEnvelope (in Envelope: IEnvelope) : IEnumElement	Returns the elements that intersect with the given envelope.
← MoveElementFromGroup (in Group: IGroupElement, in Element: IElement, in zorder: Long)	Move the element from the group to the container.
← MoveElementToGroup (in Element: IElement, in Group: IGroupElement)	Move the element from the container to the group.
← Next: IElement	Returns the next graphic in the container.
← PutElementOrder (in order: Variant)	Private order object. Used to undo ordering operations.
← Reset	Reset internal cursor so that Next returns the first element.
← SendBackward (in Elements: IEnumElement)	Move the specified elements one step closer to the bottom of the stack of elements.
← SendToBack (in Elements: IEnumElement)	Make the specified elements draw behind all other elements.
← UpdateElement (in Element: IElement)	The graphic element's properties have changed.

IGraphicsContainer provides access to the *PageLayout* object's graphic elements. Use this interface to add new elements or access existing ones. For example, a title at the top of a layout is a *TextElement* stored in the layout's graphics container.

The script below shows one method for adding a new text element onto the page layout. In this example, a *UIToolControl* is used to get a mouse down event so users can place the text element anywhere they desire on the page layout. The script will only add a new element if ArcMap is in layout view. To use this sample, paste the code into VBA. Use the Commands tab of the Customize dialog to create a new *UIToolControl*. If you use the default name *UIToolControl1*, the code will be associated with the *MouseDown* event of the tool. Add the tool to any toolbar, select the tool and click anywhere in the page layout to create a text element.

[Visual Basic 6.0]

```
Private Sub UIToolControl1_MouseDown(ByVal button As Long, ByVal shift As Long, ByVal x As Long, ByVal y As Long)
    Dim pMxDoc As IMxDocument
    Dim pPageLayout As IPageLayout
    Dim pActiveView As IActiveView
    Dim pGraphicsContainer As IGraphicsContainer
    Dim pTextElement As ITextElement
    Dim pElement As IElement
    Dim pPoint As IPoint

    Set pMxDoc = Application.Document
    Set pPageLayout = pMxDoc.PageLayout
    Set pActiveView = pPageLayout 'QI
    Set pGraphicsContainer = pPageLayout 'QI
    'Check that ArcMap is in layout view
    If Not TypeOf pMxDoc.ActiveView Is IPageLayout Then
        MsgBox "Tool works only in layout view"
        Exit Sub
    End If
    Set pTextElement = New TextElement
    Set pElement = pTextElement 'QI
    'Create a point from the x,y coordinate parameters
    Set pPoint = _
        pActiveView.ScreenDisplay.DisplayTransformation.ToMapPoint(x, y)
    pTextElement.Text = "My Map"
    pElement.Geometry = pPoint
    pGraphicsContainer.AddElement pTextElement, 0
    'Refresh only the pagelayout's graphics
```

```
pActiveView.PartialRefresh esriViewGraphics, Nothing, Nothing
End Sub
```

This script moves all the elements in the layout one inch to the right:

[Visual Basic 6.0]

```
Public Sub MoveAllElements()
    Dim pMxDoc As IMxDocument
    Dim pPageLayout As IPageLayout
    Dim pActiveView As IActiveView
    Dim pGraphicsContainer As IGraphicsContainer
    Dim pElement As IElement
    Dim pTransform2D As ITransform2D
    Set pMxDoc = Application.Document
    Set pPageLayout = pMxDoc.PageLayout
    Set pActiveView = pPageLayout 'QI
    Set pGraphicsContainer = pPageLayout 'QI
    'Loop through all the elements and move each one 1 inch
    pGraphicsContainer.Reset
    Set pElement = pGraphicsContainer.Next
    Do While Not pElement Is Nothing
        Set pTransform2D = pElement
        pTransform2D.Move 1, 0
        Set pElement = pGraphicsContainer.Next
    Loop
    'Refresh only the pagelayout's graphics
    pActiveView.PartialRefresh esriViewGraphics, Nothing, Nothing
End Sub
```

***IGraphicsContainerSelect*: Selecting graphics**

IGraphicsContainerSelect : IUnknown	Provides access to members that control graphic container selection.
<ul style="list-style-type: none"> ■ □ DominantElement: IElement ■ ElementSelectionCount: Long ■ SelectedElements: IEnumElement ■ SelectionBounds (in Display: IDisplay) : IEnvelope 	<ul style="list-style-type: none"> Dominant element. Returns the number of selected elements. Returns the selected elements. Returns the bounds of the selection.
<ul style="list-style-type: none"> ← ElementSelected (in Element: IElement) : Boolean ← SelectAllElements ← SelectedElement (in Index: Long) : IElement ← SelectElement (in Element: IElement) ← SelectElements (in Elements: IEnumElement) ← SelectionTracker (in Index: Long) : ISelectionTracker ← UnselectAllElements ← UnselectElement (in Element: IElement) ← UnselectElements (in Elements: IEnumElement) 	<ul style="list-style-type: none"> Indicates if the element is selected. Selects all elements. Returns the nth selected element. Use Selection count to get the number of selected elements. Selects the specified element. Selects the specified elements. Returns the tracker for the nth selected element. Use Selection count to get the number of selected elements. Unselects all elements. Unselects the specified element. Unselects the specified elements.

Most objects that are graphics containers, such as *PageLayout* and *Map*, implement the *IGraphicsContainerSelect* interface to expose additional members for managing their element selection. For example, *IGraphicsContainerSelect::UnselectAllElements* can be used to clear an object's graphic element selection.

The following simple VBA example returns the number of elements currently selected in the focus *Map* and the *PageLayout*:

[Visual Basic 6.0]

```
Public Sub GraphicSelectionCount()
    Dim pMxDocument As IMxDocument
    Dim pMap As IMap
    Dim pPageLayout As IPageLayout
    Dim pMapGraphicsSelect As IGraphicsContainerSelect
    Dim pPageLayoutGraphicsSelect As IGraphicsContainerSelect
    Set pMxDocument = Application.Document
    Set pMap = pMxDocument.FocusMap
    Set pPageLayout = pMxDocument.PageLayout
    Set pMapGraphicsSelect = pMap
    Set pPageLayoutGraphicsSelect = pPageLayout
    MsgBox "Selected elements in the map: " & pMapGraphicsSelect.ElementSelectionCount
End Sub
```

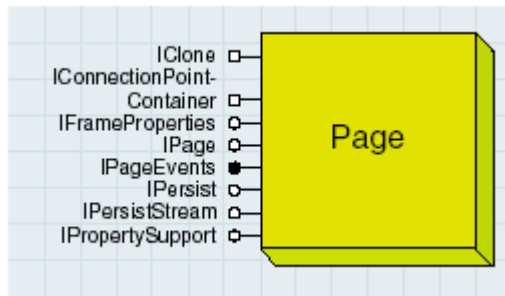


```

MsgBox "Selected elements in the page layout: " _
    & pPageLayoutGraphicsSelect.ElementSelectionCount
End Sub

```

The *Page* coclass



The *PageLayout* object automatically creates a *Page* object to manage the page of paper on which the layout resides. Aside from color, size, and orientation, the *Page* object manages additional layout properties, such as page units, border style, and printable bounds. Access the *PageLayout*'s *Page* object via *IPageLayout::Page*.

The *IPage* interface

IPage : IUnknown	Provides access to members that control the Page.
<ul style="list-style-type: none"> Background: IBackground BackgroundColor: IColor Border: IBorder DelayEvents: Boolean FormID: esriPageFormID IsPrintableAreaVisible: Boolean Orientation: Integer PageToPrinterMapping: esriPageToPrinterMapping PrintableBounds: IEnvelope StretchGraphicsWithPage: Boolean Units: esriUnits 	<ul style="list-style-type: none"> The page background. The page color. The page border. Indicates if the page stops firing IPageEvents until the flag is set to false. The Page form. Indicates if the printable area is visible. The Page orientation. 1 = portrait 2 = landscape. The page to printer mapping. The printable bounds. Indicates if graphics should stretch with the page when the page size changes. The units used for the page and all associated coordinates.
<ul style="list-style-type: none"> DrawBackground (in Display: IDisplay) DrawBorder (in Display: IDisplay) DrawPaper (in Display: IDisplay, in eraseColor: IColor) DrawPrintableArea (in Display: IDisplay) GetDeviceBounds (in Printer: IPrinter, in currentPage: Integer, in Overlap: Double, in Resolution: Integer, in deviceBounds: IEnvelope) GetPageBounds (in Printer: IPrinter, in currentPage: Integer, in Overlap: Double, in pageBounds: IEnvelope) PrinterChanged (in Printer: IPrinter) PrinterPageCount (in Printer: IPrinter, in Overlap: Double, out pageCount: Integer) PutCustomSize (in Width: Double, in Height: Double) QuerySize (out Width: Double, out Height: Double) 	<ul style="list-style-type: none"> Draw the page background. Draw the page border. Draw the paper. EraseColor is the color of the area surrounding the page. Only the area around the page is drawn in order to eliminate flashing. Use EraseColor = 0 to simply draw page. Draw the printable area. The number of printer pages spanned by the page. The number of printer pages spanned by the page. Called by PageLayout when printer changes. The number of printer pages spanned by the page. The size of the page in page units. The size of the page in page units.

IPage is the primary interface on the *Page* object. Use this interface to access all of the properties of an ArcMap page, including the page's border, background, background color, orientation, and size.

The two samples provide here change the size and color of the page:

[Visual Basic 6.0]

```

Public Sub CheckPageSize()
    Dim pMxDoc As IMxDocument
    Dim pPage As IPage
    Dim dHeight As Double

```

```

Dim dWidth As Double
Set pMxDoc = Application.Document
Set pPage = pMxDoc.PageLayout.Page
pPage.QuerySize dWidth, dHeight
If dWidth = 8.5 And dHeight = 11 Then
    pPage.PutCustomSize 5, 5
End If
End Sub

Public Sub ChangePageColor()
Dim pMxDoc As IMxDocument
Dim pPage As IPage
Dim pColor As IColor
Dim pRgbColor As IRgbColor
Set pMxDoc = Application.Document
Set pPage = pMxDoc.PageLayout.Page
Set pRgbColor = New RgbColor
pRgbColor.Blue = 204
pRgbColor.Red = 255
pRgbColor.Green = 255
pPage.BackgroundColor = pRgbColor
End Sub

```

Standard page sizes

Enumeration <i>esriPageFormID</i>	Forms support in Page.
0 - <i>esriPageFormLetter</i>	Letter - 8.5in x 11in.
1 - <i>esriPageFormLegal</i>	Legal - 8.5in x 14in.
2 - <i>esriPageFormTabloid</i>	Tabloid - 11in x 17in.
3 - <i>esriPageFormC</i>	C - 17in x 22in.
4 - <i>esriPageFormD</i>	D - 22in x 34in.
5 - <i>esriPageFormE</i>	E - 34in x 44in.
6 - <i>esriPageFormA5</i>	Metric A5 - 148mm x 210mm.
7 - <i>esriPageFormA4</i>	Metric A4 - 210mm x 297mm.
8 - <i>esriPageFormA3</i>	Metric A3 - 297mm x 420mm.
9 - <i>esriPageFormA2</i>	Metric A2 - 420mm x 594mm.
10 - <i>esriPageFormA1</i>	Metric A1 - 594mm x 841mm.
11 - <i>esriPageFormA0</i>	Metric A0 - 841mm x 1189mm.
12 - <i>esriPageFormCUSTOM</i>	Custom Page Size.
13 - <i>esriPageFormSameAsPrinter</i>	Page Form same as Printer Form.

The *esriPageFormID* enumeration provides a convenient list of preselected page sizes for use by the *Page* object. For example, to change the layout to standard legal page size, pass in *esriPageFormLegal* to *IPage::FormID*. This is much quicker than setting a custom size with *IPage::PutCustomSize*.

One important element in this enumeration is *esriPageFormSameAsPrinter*. When the *FormID* property has been set to this element, the layout's page size mimics the page size of the printer; whenever the printer page size changes, the layout's page size changes to match it. You can see this behavior in the ArcMap application on the Page Setup dialog box accessed from the File menu. Click the File menu and click Page Setup. If the Same as Printer check box is checked, the map setup will change to reflect any changes to the printer setup.

This sample uses the *esriPageFormID* enumeration to quickly change the page size. It may be quite useful if you just used the previous code sample to change the page's size and color.

[Visual Basic 6.0]

```

Public Sub SetLegalPageSize()
Dim pMxDoc As IMxDocument
Dim pPageLayout As IPageLayout
Dim pPage As IPage
Dim x As Double, y As Double
Set pMxDoc = Application.Document
Set pPageLayout = pMxDoc.PageLayout
Set pPage = pPageLayout.Page
pPage.FormID = esriPageFormLegal
pPage.QuerySize x, y
MsgBox "The page size is now: " & x & " " & y
End Sub

```

Mapping the page size to the printer

Enumeration esriPageToPrinterMapping

- 0 - esriPageMappingCrop
- 1 - esriPageMappingScale
- 2 - esriPageMappingTile

Page to Printer Mapping.

- Crop Page to Printer.
- Scale Page to Printer.
- Tile Page to Printer.

The *esriPageToPrinterMapping* enumeration tells the *Page* what to do when the layout's page size does not match the printer's page size. This is often the case when *IPage::FormID* is set to something other than *esriPageFormSameAsPrinter*. By default, ArcMap crops the page, but you can choose to either scale the page or tile it. In the ArcMap application, you can see these choices on the Print dialog box.

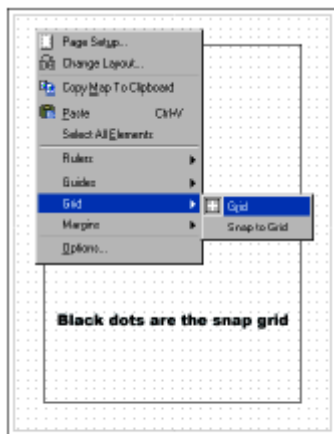
Page events**IPageEvents : IUnknown**

- ← PageColorChanged
- ← PageMarginsChanged
- ← PageSizeChanged
- ← PageUnitsChanged

Provides access to events that occur when the Page changes.

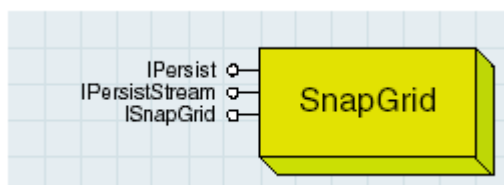
- Fired when the page color changes.
- Fired when the page margins change.
- Fired when the page size changes.
- Fired when the units used by the page changes.

The *Page* object is the event source for page events. *Page* events are fired by the *Page* object to notify all clients that certain aspects of the page have changed. The page events are grouped under the *IPageEvents* interface and are *PageColorChanged*, *PageMarginsChanged*, *PageSizeChanged*, and *PageUnitsChanged*. Within ArcMap, there is only one client—the *PageLayout* object—listening for these events. The *PageLayout* object listens for these events so it can modify its layout according to changes made to its page. For example, when the page units are changed, the page layout needs to update its transformation, update the snap tolerance and snap grid, update its snap guides, and convert its graphics to the new units.

The snap grid

This image shows the snap grid.

The layout view supports a snap grid, which is a grid of reference points on the layout used to help position elements. The grid may be used as a visual indicator of size and position, or it may be used to snap elements into position.



In layout view, right-click the screen and click Grid. This lets you show or hide the snap grid, as well as enable or disable snapping to the grid. The *SnapGrid* object represents the snap grid. Although this object is creatable, there is generally no need to create one as the *PageLayout* object automatically creates one when it is created. Use *IPageLayout::SnapGrid* to get a reference to the snap grid currently associated with the layout view.

For information about enabling and disabling grid snapping, see the section on graphic snap agents [below](#).

The *SnapGrid* implements *IPersist* and *IPersistStream* to save the object's current settings in the current

map document.

ISnapGrid : IUnknown	Provides access to members that control the Snapping grid.
HorizontalSpacing: Double	<i>The horizontal distance between grid points.</i>
IsVisible: Boolean	<i>Indicates if the snapping grid is visible.</i>
VerticalSpacing: Double	<i>The vertical distance between grid points.</i>
Draw (in Display: IDisplay, in Page: IPage)	<i>Draw the grid.</i>

The primary interface on the *SnapGrid* object is *ISnapGrid*. Use this interface to change the grid's horizontal and vertical spacing and control whether or not the grid is visible. The sample below changes the snap grid's vertical and horizontal spacing to 0.5 inches and ensures the grid is visible.

[Visual Basic 6.0]

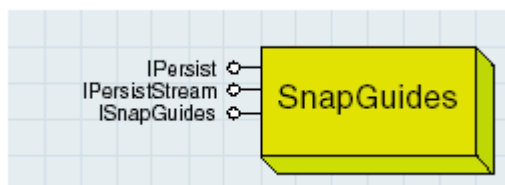
```
Public Sub SnapGrid()
    Dim pMxDoc As IMxDocument
    Dim pSnapGrid As ISnapGrid
    Dim pActiveView As IActiveView
    Set pMxDoc = Application.Document
    Set pSnapGrid = pMxDoc.PageLayout.SnapGrid
    pSnapGrid.HorizontalSpacing = 0.5
    pSnapGrid.VerticalSpacing = 0.5
    pSnapGrid.IsVisible = True
    Set pActiveView = pMxDoc.PageLayout
    pActiveView.Refresh
End Sub
```

Snap guides



This image shows a vertical and a horizontal snap guide added to the layout.

You can use rulers, guides, and grids in layout view to align elements on the page.



The *PageLayout* object has two *SnapGuides* objects, one for managing horizontal guides, and one for managing vertical guides. Use *IPageLayout::VerticalSnapGuides* or *IPageLayout::HorizontalSnapguides* to obtain a reference to the desired *SnapGuides* object.

Each *SnapGuides* object manages an internal collection of individual guides. For example, the *SnapGuides* object that represents the horizontal snap guides may contain 10 individual guides.

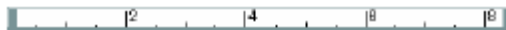
ISnapGuides : IUnknown	Provides access to members that control the Snapping guides.
<ul style="list-style-type: none"> ■ AreVisible: Boolean ■ DrawLevel: tagesriViewDrawPhase ■ Guide (in idx: Long) : Double ■ GuideCount: Long 	<ul style="list-style-type: none"> <i>Indicates if snapping guides are visible.</i> <i>Level where guides are drawn.</i> <i>The nth guide. The position is specified in page units.</i> <i>The number of guides.</i>
<ul style="list-style-type: none"> ← AddGuide (in pos: Double) ← Draw (in Display: IDisplay, in isHorizontal: Boolean) ← DrawHighlight (in Display: IDisplay, in isHorizontal: Boolean) ← RemoveAllGuides ← RemoveGuide (in idx: Long) 	<ul style="list-style-type: none"> <i>Adds a guide at the specified position. The position is specified in page units.</i> <i>Draw a fine line showing exactly where objects will snap.</i> <i>Draw a highlight around the snap line for a nice visual effect.</i> <i>Removes all the guides.</i> <i>Removes the nth guide.</i>

Use *ISnapGuides* to add a new guide, remove a guide, and turn the visibility of the guides on or off. The sample below adds a new horizontal guide 5 inches from the bottom of the page and then turns on the horizontal guides' visibility if they are turned off.

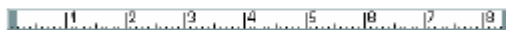
[Visual Basic 6.0]

```
Public Sub AddHorizontalSnapGuide()
'Add a horizontal snap guide 5 inches up the page
    Dim pMxDoc As IMxDocument
    Dim pHorizontalSnapGuides As ISnapGuides
    Dim pActiveView As IActiveView
    Set pMxDoc = Application.Document
    Set pHorizontalSnapGuides = pMxDoc.PageLayout.HorizontalSnapGuides
    pHorizontalSnapGuides.AddGuide 5
    If Not pHorizontalSnapGuides.AreVisible Then
        pHorizontalSnapGuides.AreVisible = True
        Set pActiveView = pMxDoc.PageLayout
        pActiveView.Refresh
    End If
End Sub
```

Rulers settings

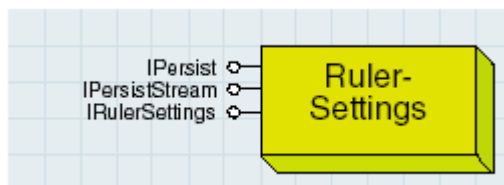


This image shows the horizontal ruler with the SmallestDivision property set to 2.



This image shows the same ruler again but with the SmallestDivision property set to 0.1. Notice that there are now 10 markings between each inch.

Rulers show the size of a page and elements on the final printed map.



The *PageLayout* object has a *RulerSettings* object that manages the ruler settings. Although this object is cocreatable, there is generally no need to create one because the *PageLayout* object automatically instantiates one when it is created. Use *IPageLayout::RulerSettings* to get a reference to the *RulerSettings* object currently associated with the layout view.

IRulerSettings : IUnknown	Provides access to members that control Ruler setup.
<ul style="list-style-type: none"> ■ SmallestDivision: Double 	<ul style="list-style-type: none"> <i>The size of the smallest ruler division. The size is in page units.</i>

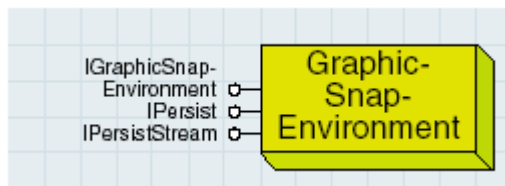
The *IRulerSettings* interface only has one property, *SmallestDivision*. This property controls the size of the smallest ruler division in page units. For example, if the page size is 8.5 by 11 inches and the *SmallestDivision* is set to 2, the rulers in layout view will read off every 2 inches. If the property is set to .1, the rulers will read off every 1/10 of an inch.

[Visual Basic 6.0]

```
Public Sub ChangeRulerSettings()
    Dim pMxDoc As IMxDocument
    Dim pRulerSettings As IRulerSettings
    Set pMxDoc = Application.Document
    Set pRulerSettings = pMxDoc.PageLayout.RulerSettings
    pRulerSettings.SmallestDivision = 2
    pMxDoc.ActiveView.Refresh
End Sub
```

The graphic snap environment

The graphic snap environment controls which graphic snap agents are active, the order in which they are called, and the snap tolerance.



To aid in aligning and positioning elements on a page, the layout view supports element snapping. Elements may be snapped to the snap grid, rulers, guides, and margins. Snapping is performed by a combination of efforts between snapping target objects and snap agents. The snap agents attempt to move a graphic to a position on a snapping target object. The *PageLayout* object manages the snap agents, snapping target objects, and the snapping environment.

The *GraphicSnapEnvironment* object manages the graphic snap agents. This object is cocreatable, but typically this is not necessary because *PageLayout* object automatically creates the object when it itself is created. The *PageLayout* actually aggregates a *GraphicSnapEnvironment* object, making it part of the *PageLayout* object.

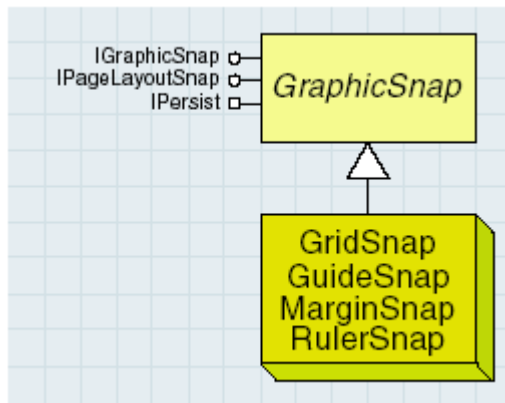
To get a reference to the *GraphicSnapEnvironment* associated with the page layout, simply perform a query interface from any of the other interfaces on *PageLayout*, such as *IPageLayout*.

IGraphicSnapEnvironment : IUnknown	Provides access to members that control the Collection of snap agents used for snapping graphics.
<ul style="list-style-type: none"> ■ SnapAgent (in Index: Long) : IGraphicSnap ■ SnapAgentCount: Long ■ SnapAgentOrder: IArray ■ SnapTolerance: Double 	<p>Get a snap agent. The index argument is zero based.</p> <p>The number of snap agents.</p> <p>An array of IDs indicating how agents should be ordered.</p> <p>The snap tolerance in page units.</p>
<ul style="list-style-type: none"> ← AddSnapAgent (in SnapAgent: IGraphicSnap) ← ClearSnapAgents ← DeleteSnapAgent (in SnapAgent: IGraphicSnap) ← SnapShape (in Shape: IGeometry) 	<p>Add a new snap agent to the environment.</p> <p>Remove all snap agents.</p> <p>Remove specified snap agent from the environment.</p> <p>Snap the shape using the agents in the environment.</p>

Use the *IGraphicSnapEnvironment* interface to add or delete snap agents and to snap a graphic into place with *SnapShape*. The *SnapShape* method calls each snap agent's snap method until one of them returns True, indicating that they have moved the graphic. When a snap agent returns True, no other snap agents are called. You can also use the *SnapAgentOrder* property on this interface to control in which order the snap agents are called. With this interface, you can establish a snap agent priority—for example, you may decide snapping to the snap grid is more important than snapping to the page margins.

Graphic snap coclasses

The grid snap moves graphics to the snap grid. The guide snap moves graphics to the horizontal and vertical guides. The margin snap snaps graphics to the layouts printable bounds. The ruler snap snaps graphics to the rulers.



Rulers, guides, and grids are layout objects that aid in aligning elements on a page. However, these objects are only half the story—there are also snap agents that snap to them. Layout snap agents include *GridSnap*, *GuideSnap*, *MarginSnap*, and *RulerSnap*. There is a one-to-one correlation between the snap agents and the objects to which they snap. For example, the *GridSnap* snap agent attempts to snap graphic elements to the snap grid created by the *SnapGrid* object. The exception is the *MarginSnap* snap agent, which simply snaps to the layout's printable bounds (*IPage::PrintableBounds*).

Graphics are snapped into place by calling *IGraphicSnapEnvironment::SnapShape* on the *PageLayout* object. *SnapShape* in turn calls *IGraphicSnap::SnapX* and *IGraphicSnap::SnapY* on each active snap agent (in the order specified by *IGraphicSnapEnvironment::SnapOrder*) until one of the snap agents returns True, indicating that a new point has been found that meets the criteria of the snap agent. *SnapX* and *SnapY* are separate calls because some agents, such as guides, may only act in one direction.

GraphicSnap is an abstract class with the interface *IGraphicSnap*, which all graphic snap agents implement.

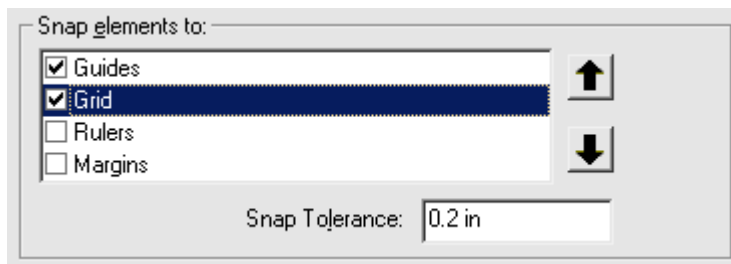
In ArcMap, a guide snap agent is automatically created and then snaps to vertical and horizontal snap guides. There is no need to create more than one type of snap agent. In ArcMap, you can access the snapping environment and snap agents by right-clicking in the layout view and clicking Options. On the Layout View tab, you can turn snap agents on or off, control the snap agent order, and set the snap tolerance.

IGraphicSnap : IUnknown	Provides access to members that control snapping graphics.
← Name: String	The name of the snap agent.
← SnapX (in Shape: IGeometry, in Tolerance: Double) : Boolean	Indicates if the point is snapped in the horizontal direction.
← SnapY (in Shape: IGeometry, in Tolerance: Double) : Boolean	Indicates if the point is snapped in the vertical direction.

All graphic snap agents implement the *IGraphicSnap* interface. This interface only has three members: *Name*, *SnapX*, and *SnapY*. *SnapX* and *SnapY* are unique and are used to determine if a graphic can be snapped. For example, the *GridSnap* agent's implementation of *SnapX* for polygon graphics checks if either the Xmin or Xmax of the graphics bounding rectangle is within snap tolerance of the snap grid. If either is, the graphic is moved the calculated distance between the two. *SnapX* and *SnapY* always return a Boolean, indicating whether or not the graphic was snapped. If any snap agent returns True, no other snap agents are called.

IPageLayoutSnap : IGraphicSnap	Provides access to members that control snap agents that are used with PageLayout.
← PageLayout: IPageLayout	Sets the PageLayout that this snap agent is associated with.

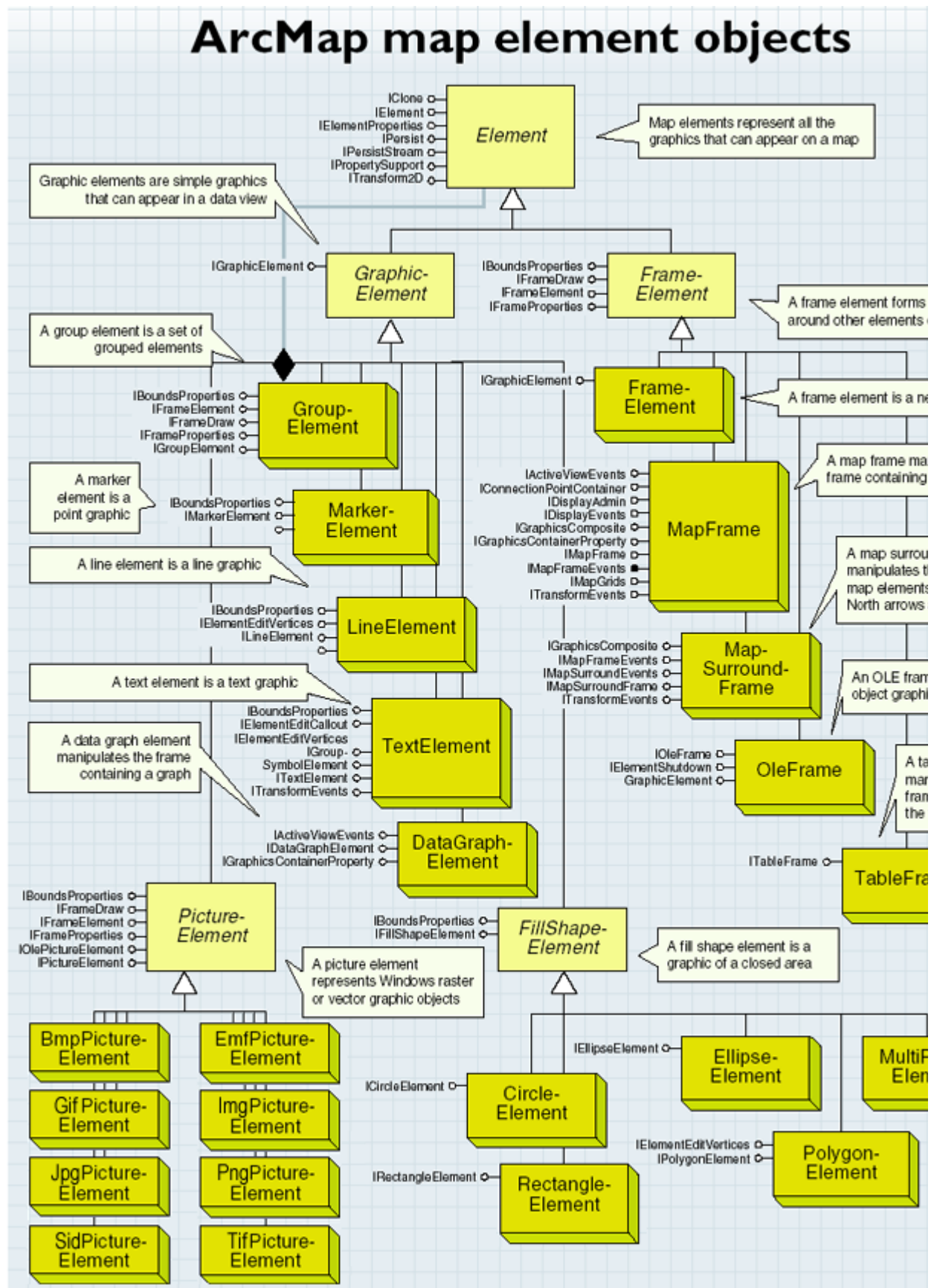
This interface is used to tie the snap agents to the *PageLayout* object. If this property is not set, the graphic snap agents will not work properly. Because *IPageLayoutSnap* inherits from *IGraphicSnap*, all the methods on *IGraphicSnap* are directly available on *IPageLayoutSnap*. The following sample demonstrates how a grid snap agent can be added to the layout. After you have run this code right click in the Layout and open the Options dialog. On the Layout View tab, the Snap elements to Grid option will be checked.



[Visual Basic 6.0]

```
Public Sub AddGridSnapAgent()  
    Dim pMxDoc As IMxDocument  
    Dim pPageLayout As IPageLayout  
    Dim pSnapEnv As IGraphicSnapEnvironment  
    Dim pPageLayoutSnap As IPageLayoutSnap  
    Set pMxDoc = Application.Document  
    Set pPageLayout = pMxDoc.PageLayout  
    Set pSnapEnv = pPageLayout  
    Set pPageLayoutSnap = New GridSnap  
    pPageLayoutSnap.PageLayout = pPageLayout  
    pSnapEnv.AddSnapAgent pPageLayoutSnap  
End Sub
```

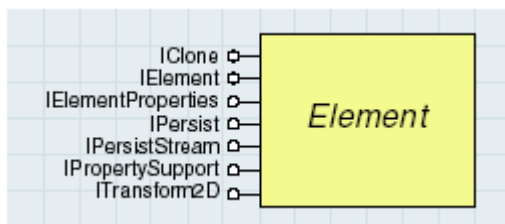
Map elements



Elements are used in the map or the page layout to display basic shapes, text labels, etc. They can also be used in the page layout to display marginalia. They are very useful to display on or around the map this information that is not directly conveyed by the geographic features.

A map layout and a data frame can both contain elements, but elements are most commonly manipulated as part of a map layout. Elements can basically be thought of as the non feature-based components of a map. The list of supported elements includes *FrameElements*, which hold maps; *MapSurroundFrames*,

which hold North arrows, scale bar, and so on; and *GraphicElements*, which hold text, line, point, fillshape, and picture elements.



Element is the abstract class on which all graphic elements and frames are based.

Elements are commonly accessed through the *IGraphicsContainer* interface implemented by the *Map* and *PageLayout* objects. Through this interface you can add, delete, update, and retrieve the individual elements within a *Map* or *PageLayout*. Use the *GroupElement* object to combine multiple elements into a single unit for manipulation by the user.

IElement : IUnknown	Provides access to members that control the Element.
<ul style="list-style-type: none"> ■ Geometry: IGeometry ■ Locked: Boolean ■ SelectionTracker: ISelectionTracker 	<ul style="list-style-type: none"> Shape of the element as a geometry. Indicates if the element is in a read-only state. Selection tracker used by this element.
<ul style="list-style-type: none"> ← Activate (in Display: IDisplay) ← Deactivate ← Draw (in Display: IDisplay, in trackCancel: ITrackCancel) ← HitTest (in X: Double, in Y: Double, in Tolerance: Double) : Boolean ← QueryBounds (in Display: IDisplay, in Bounds: IEnvelope) ← QueryOutline (in Display: IDisplay, in Outline: IPolygon) 	<ul style="list-style-type: none"> Prepare to display graphic on screen. ActiveView that graphics are displayed on is no longer visible. Draws the element into the given display object. Indicates if the given x and y coordinates are contained by the element. Bounds of the element taking symbology into consideration. Bounds of the element taking symbology into consideration.

IElement is the generic interface implemented by all graphic elements and frames. Most methods that return graphics (various methods and properties of *IGraphicsContainer* and *IGraphicsContainerSelect*) return them as generic *IElement* objects. *IElement* gives the programmer access to the geometry of the object and employs methods for querying the object and drawing it. It is the programmer's responsibility to determine what type of object is hosting the *IElement* interface by performing a QI. In VB, check the elements in a page layout for *PolygonElements* in the following manner:

[Visual Basic 6.0]

```

Sub CountPolygonElements()
    Dim pDoc As IMxDocument
    Dim pPageLayout As IPageLayout
    Dim pContainer As IGraphicsContainer
    Dim pElement As IElement
    Dim i As Integer
    Set pDoc = ThisDocument
    Set pPageLayout = pDoc.PageLayout
    Set pContainer = pPageLayout
    pContainer.Reset
    Set pElement = pContainer.Next
    Do While Not pElement Is Nothing
        If TypeOf pElement Is IPolygonElement Then
            i = i + 1
        End If
        Set pElement = pContainer.Next
    Loop
    MsgBox "The Page Layout contains " & i & " PolygonElement(s)"
End Sub
  
```

The *SelectionTracker* property will return an *ISelectionTracker*, which can be used to reshape the element. Reshaping of elements is done via handles around the edges of the element. *QueryBounds* and *QueryOutline* both require instantiated objects to be passed in. The results of each will be the same for line and point elements, but will vary for polygon elements (*QueryBounds* returns the bounding box, while *QueryOutline* will return an outline of the element).

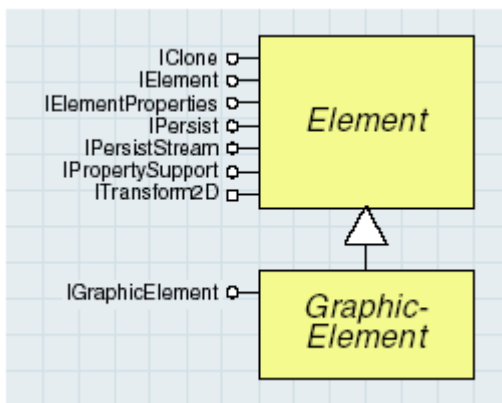
IElementProperties : IUnknown	Provides access to members that control the Element Properties.
<ul style="list-style-type: none"> AutoTransform: Boolean CustomProperty: Variant Name: String Type: String 	<p>Indicates if transform is applied to symbols and other parts of element. False = only apply transform to geometry.</p> <p>Custom property.</p> <p>Name of the element.</p> <p>Type of the element.</p>

IElementProperties is a generic interface implemented by all graphic elements and frames. This interface allows the developer to attach custom properties to an element. The *Name* and *Type* properties allow the developer to categorize their custom properties. Use the *IElementProperties2* interface instead to access the reference scale of the element.

AutoTransform is a Boolean value that indicates whether internal settings should be transformed along with the element's geometry when a transform is applied via *ITransform2D*. For instance, if you have a point element and you rotate it around a central location (the anchor point of the rotation being different from the point element itself), then the *AutoTransform* property is used to determine whether the orientation of the symbol associated to the element should also be rotated by the same amount.

Graphic elements

Descending from the *Element* abstract class, *GraphicElement* objects are elements that work in both a data frame and a map layout. This category of elements includes text, lines, points, polygons, and pictures.



Graphic elements are added to a data frame or map to highlight areas or provide detail beyond that of the geographic features. The process of redlining (marking areas for correction or notification) can be done by adding graphic elements to the map. Annotation, which is used to label features, is unique in that it is both a geographic feature and a graphic element (specifically a *TextElement*). Annotation is added to the map based on attribute values or other text strings.

IGraphicElement : IUnknown	Provides access to members that control the Graphic Element object.
<ul style="list-style-type: none"> SpatialReference: ISpatialReference 	<p>Spatial reference of the map.</p>

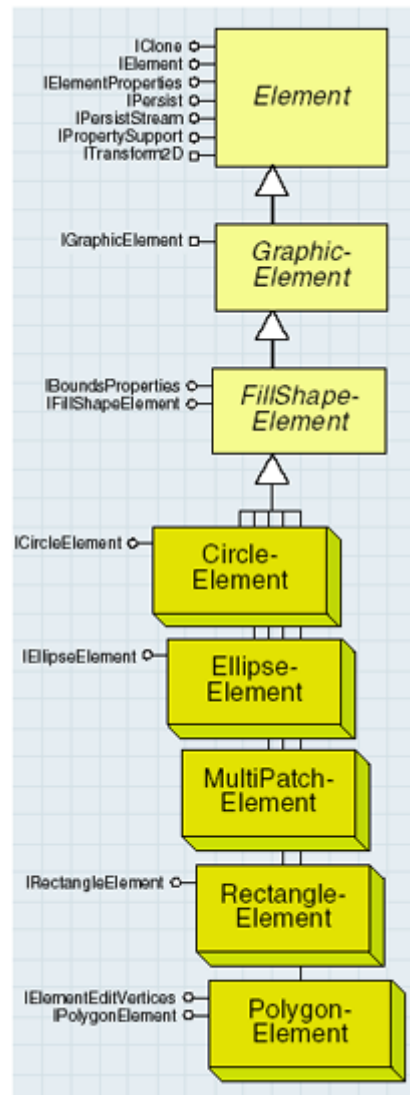
The *IGraphicElement* interface is a generic interface implemented by all graphic elements. This interface provides access to the spatial reference of the element.

ITransform2D : IUnknown	Provides access to members that supply an object with Euclidean 2D transformation capabilities.
<ul style="list-style-type: none"> Move (dx: Double, dy: Double) MoveVector (v: ILine) Rotate (Origin: IPoint, RotationAngle: Double) Scale (Origin: IPoint, sx: Double, sy: Double) Transform (Direction: ITransformDirection, Transformation: ITransformation) 	<p>Moves the object dx units horizontally and dy units vertically.</p> <p>Moves the object defined by a 2D displacement vector.</p> <p>Rotates the object about the specified origin point through rotationAngle radians.</p> <p>Scales the object about the specified origin point a factor of sx horizontally and sy vertically.</p> <p>Applies an arbitrary transformation.</p>

The *ITransform2D* interface is implemented by elements and basic geometries (points, polylines, and so on) to aid in the repositioning of objects. This interface allows elements and geometries to be moved,

rotated, scaled, and transformed to new locations. It is implemented for graphic elements so that they can move along when the geometry of the features to which they are linked is modified.

FillShape elements



The *FillShapeElement* abstract class is a type of *Element*, but it is also an *CircleElement*, *EllipseElement*, *PolygonElement*, and *RectangleElement*. Each element represents a two-dimensional, closed-area graphic.

IFillShapeElement : IUnknown	Provides access to members of an element.
<ul style="list-style-type: none"> Symbol: IFillSymbol 	Fill symbol this element uses to display.

IFillShapeElement is a generic interface supported by all *FillShapeElement* access to the symbology used in displaying the element.

IPropertySupport : IUnknown	Provides access to members of an object.
<ul style="list-style-type: none"> Current (in pUnk: IUnknown Pointer) : IUnknown Pointer Applies (in pUnk: IUnknown Pointer) : Boolean Apply (in NewObject: IUnknown Pointer) : IUnknown Pointer CanApply (in pUnk: IUnknown Pointer) : Boolean 	<p>The object currently being used.</p> <p>Indicates if the receiver can apply the given property to the object.</p> <p>Indicates if the receiver can apply the property at the current moment.</p>

IPropertySupport is the interface implemented by *Elements* and various *Layers* (*DimensionLayer*, *FeatureLayer*, *TinEdgeRenderer*, and others) to provide access to the object. The interface determines whether a certain property can be applied to that object when appropriate.

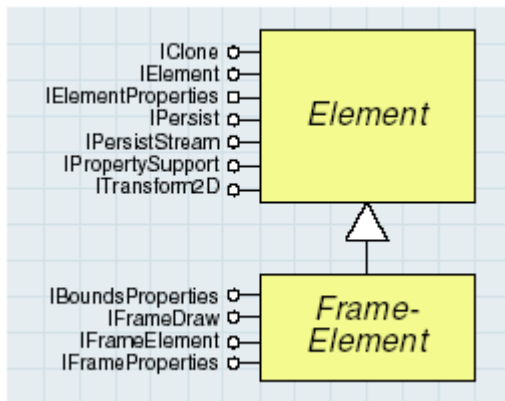
Applies indicates whether an object can be applied at all, while *CanApply* can be applied at that particular moment (whether or not the object is currently being used). *Current* will return the current object of the specified type. For instance, for its current *IColor* property.

There are many types of graphic elements. The following table is a list of each element type and a description of it. Click on the links for help on the corresponding elements.

Graphic Element	Description
GroupElement	The group graphic element to display a group of graphic elements.
MarkerElement	The graphic element to display markers (point symbols).
LineElement	The graphic element to display lines.
TextElement	The graphic element to display text.
DataGraphElement	The graphic element to display a graph.
CircleElement	The graphic element to display circles.
EllipseElement	The graphic element to display Ellipses.
PolygonElement	The graphic element to display polygons.
RectangleElement	The graphic element to display rectangles.
MultiPatchElement	The graphic element to display multipatch 3D elements. <i>MultiPatchElements</i> are elements that use a multipatch geometry and they are used in <i>ArcScene</i> . A multipatch is a series of three-dimensional surfaces that are represented as groups of geometries.

ParagraphTextElement	The graphic element to display text which flows into an area geometry.
Text3DElement	The graphic element to display 3D text
BmpPictureElement	The graphic element to display BMP pictures.
EmfPictureElement	The graphic element to display Emf pictures.
GifPictureElement	The graphic element to display GIF pictures.
ImgPictureElement	The graphic element to display IMG pictures.
InkGraphic	The graphic element to display Ink Graphic Objects (Tablet PC Inks graphics).
JpgPictureElement	The graphic element to display JPG pictures.
PngPictureElement	The graphic element to display PNG pictures.
SidPictureElement	The graphic element to display SID pPictures.
TifPictureElement	The graphic element to display TIF pictures.

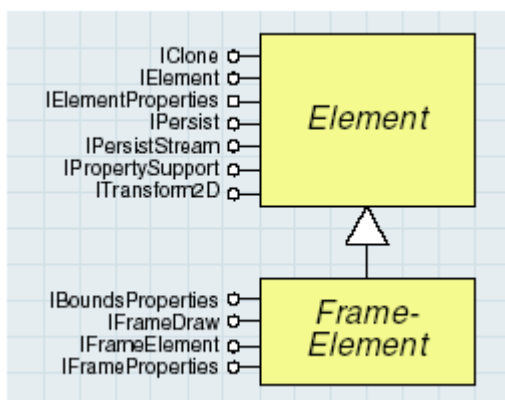
Frame elements



FrameElement is the abstract class on which all frame element objects are based.

The *FrameElement* types include *FrameElement* (holds point, line, and polygon graphics), *OleFrame* (holds OLE objects such as Word documents), *MapFrame* (holds maps), and *MapSurroundFrame* (holds North arrows, scale bar, legends, and other map primitives).

FrameElements contain other map elements—they serve as the background and border to these elements. The *MapFrame* element holds a map and allows the programmer access to that map along with the background and border properties of the container holding that map within the layout.

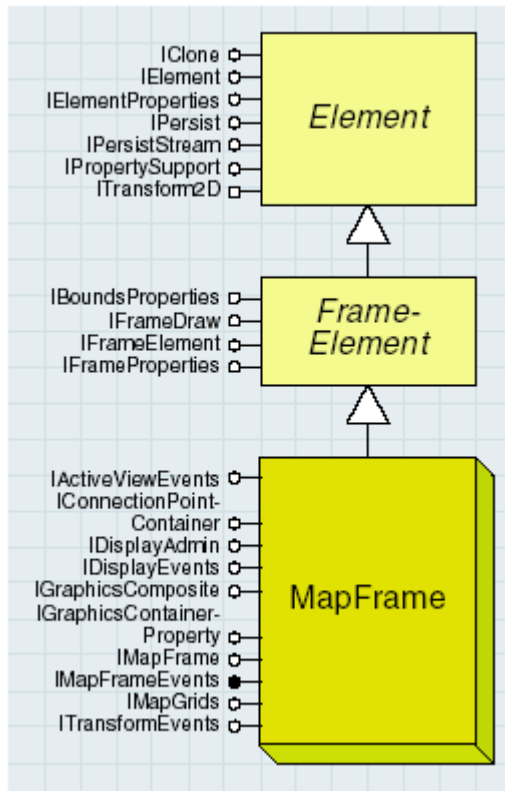


The *IFrameElement* interface is a generic interface for manipulating the properties of the frame itself (not the object within the frame). This interface provides access to the background and border properties of the frame, as well as access to the object within the frame.

The *Object* property returns the object within the frame, but it returns it as a variant. The programmer is required to determine what type of object it is. To get an *IMap* object, first determine if the *FrameElement* supports *IMapFrame*, then use the *Map* property of that interface.

Frame decorations are used to determine how frame elements are displayed. You might use a frame decoration to alter the background of an active view, add a shadow to a group of graphic elements, or draw a neatline around a map. See the [IFrameDecoration](#) interface documentation.

Map frames



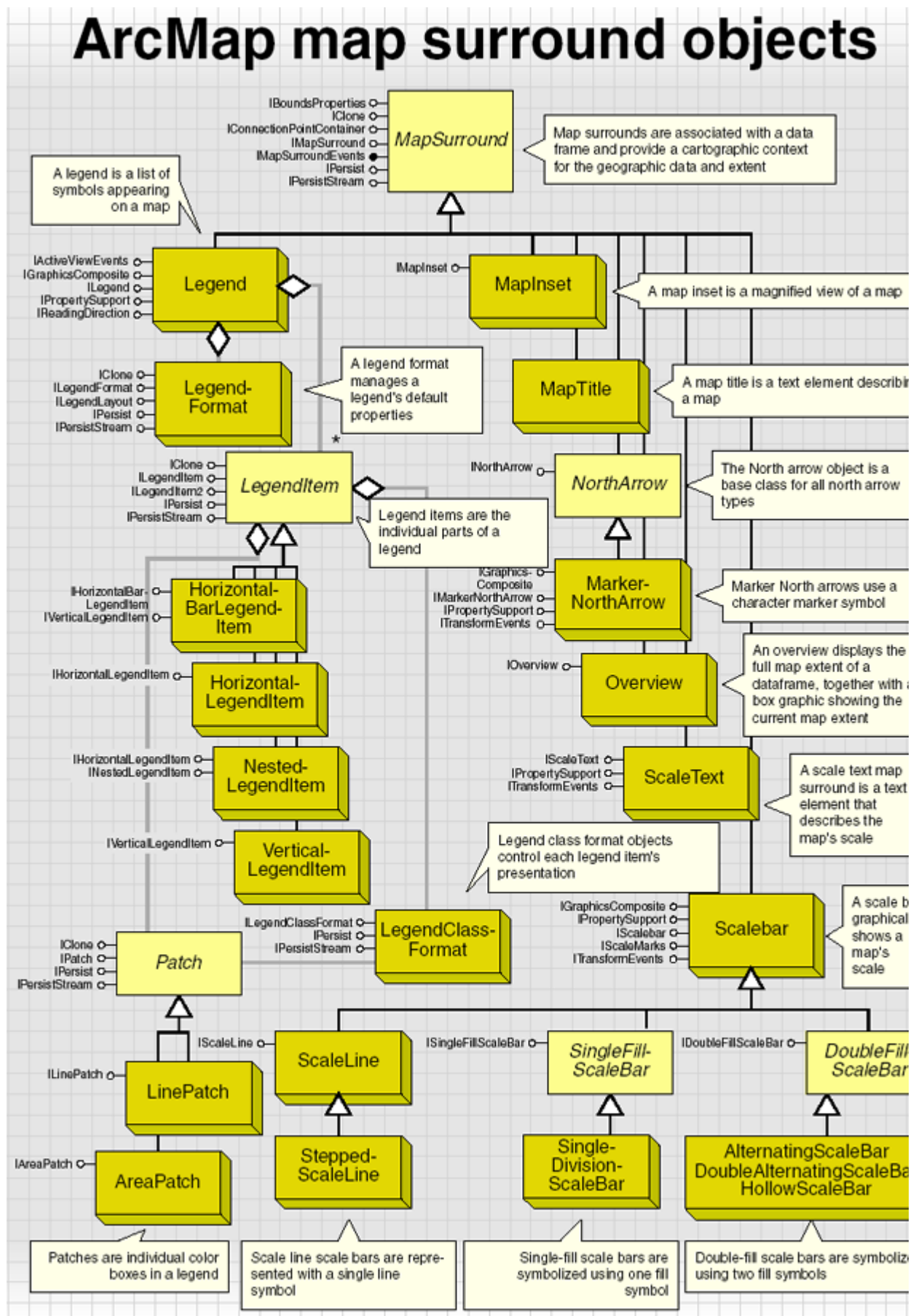
MapFrame objects are unique among the other frames and elements because they support events (*MapSurroundFrames* also support events) and reference grids. The *MapFrameResized* event is supported through *IMapFrameEvents* to allow for the updating of map grids (graticules) when the frame is resized. *Map* grids are only supported through the *MapFrame*, not on the map itself.

Check an element for the implementation of *IMapFrame* to determine if it is a *MapFrame* object.

IMapFrame : IFrameElement	Provides access to the members that control the map element object.
<ul style="list-style-type: none"> ■ Container: IGraphicsContainer ■ ExtentType: esriExtentTypeEnum ■ LocatorRectangleCount: Long ■ Map: IMap ■ MapBounds: IEnvelope ■ MapScale: Double 	<p>The frame's container.</p> <p>The way in which the map extent of the frame is specified.</p> <p>The number of locator rectangles.</p> <p>The associated map.</p> <p>The bounds of the map displayed by the frame.</p> <p>The scale at which the map should be displayed.</p>
<ul style="list-style-type: none"> ← AddLocatorRectangle (in Locator: ILocatorRectangle) ← CreateSurroundFrame (in CLSID: IUID, in optionalStyle: IMapSurround) : IMapSurroundFrame ← LocatorRectangle (in Index: Long) : ILocatorRectangle ← RemoveAllLocatorRectangles ← RemoveLocatorRectangle (in Locator: ILocatorRectangle) 	<p>Add a new locator rectangle to the data frame.</p> <p>Returns the map surround frame element of the type given in <i>clsid</i>. An optional style object may be specified.</p> <p>Returns the locator rectangle at the specified index.</p> <p>Remove all the locator rectangles from the data frame.</p> <p>Remove a locator rectangle from the data frame.</p>

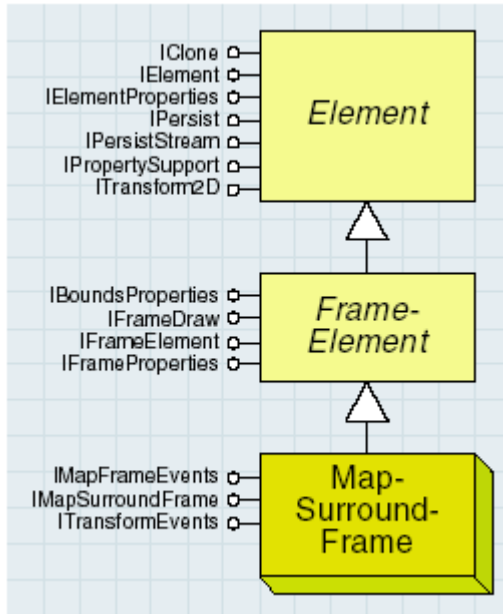
IMapFrame is the interface implemented only by the *MapFrame* coclass. The interface provides access to the map within the frame and also has the ability to create locator rectangles outlining the areas covered by other data frames. Among other things, locator rectangles can be used to highlight inset areas.

Map surrounds



Map surrounds are specific types of elements that are associated with a *Map* object. A good example of a map surround and its capabilities is the North arrow. North arrows are built as map surrounds so that they can respond to map rotation—when a map is rotated, its North arrow is rotated the same amount.

See the [Add Map Surrounds sample](#) for more about adding a legend and north arrow to the page layout.



In ArcMap, map surrounds are always contained by a *MapSurroundFrame* object—a type of element. *MapSurroundFrames* are similar to *MapFrames*, which house a *Map* object, in that the *PageLayout* object manages both of them. In fact, the *PageLayout* manages all frame objects. Each *MapSurroundFrame* is also related to a *MapFrame*; if a *MapFrame* is deleted, all of its *MapSurroundFrames* are deleted as well. Map surrounds are placed on the layout, not in a *Map*'s graphics layer.

Map surrounds can be moved anywhere on the layout, not just within the confines of a map frame. Because map surrounds are directly associated with a *Map*, the *Map* has a shortcut to all the map surrounds associated with it, *IMap::MapSurrounds*. This member, along with *IMap::MapSurroundCount*, allows you to loop through all of the available map surrounds for a given *Map* object.

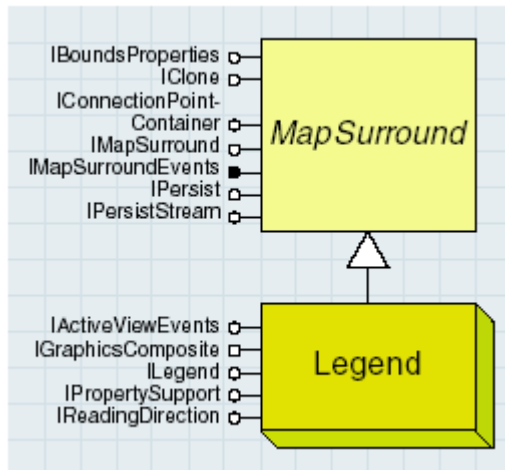
IMapSurround : IUnknown	Provides access to members that control the map surround.
<ul style="list-style-type: none"> Icon: Long Map: IMap Name: String 	<ul style="list-style-type: none"> Icon used to represent the map surround. The parent map. Name of the map surround.
<ul style="list-style-type: none"> DelayEvents (in delay: Boolean) Draw (in Display: IDisplay, in trackCancel: ITrackCancel, in Bounds: IEnvelope) FitToBounds (in Display: IDisplay, in Bounds: IEnvelope, out Changed: Boolean) QueryBounds (in Display: IDisplay, in oldBounds: IEnvelope, newBounds: IEnvelope) Refresh 	<ul style="list-style-type: none"> Used to batch operations together to minimize notifications. Draws the map surround into the specified display bounds. Adjusts the map surround to fit the bounds. The changed argument indicates whether the size of the map surround was changed. Returns the bounds of the map surround. Makes sure the latest updates are reflected the next time the Map Surround is drawn.

All map surrounds implement the *IMapSurround* interface. This interface provides all the common functionality between all map surrounds. Use this interface to access the name of a particular map surround and the associated map. This interface also has methods for determining a surround's size and changing it.

IMapSurroundEvents : IUnknown	Provides access to events that occur when the state of the map surrounds changes.
<ul style="list-style-type: none"> AfterDraw (in Display: IDisplay) BeforeDraw (in Display: IDisplay) ContentsChanged 	<ul style="list-style-type: none"> Fired after drawing completes. Fired before drawing starts. Fired when the contents of the map surround changes.

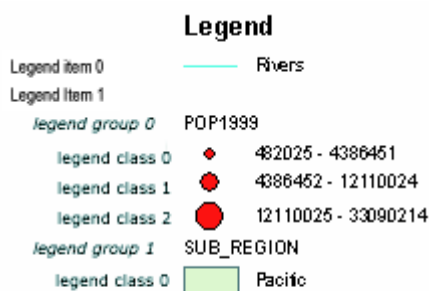
IMapSurroundsEvents is the outbound interface for all map surround objects. This interface allows you to draw a map surround in a window. The events let the window know when to redraw.

Legends



The *Legend* coclass is a complex *MapSurround* object because it relies on several other objects to create a legend.

The different elements comprised in a legend are provided by the renderers used to display the different map layers. Each renderer corresponds to a *LegendItem* in the legend. Each *LegendItem* contributes one or more *LegendGroup* objects to the legend. The number of legend groups depends on the renderer's implementation; the Multiple Attributes renderer for instance may provide up to three legend groups.



This image maps ArcMap layer renderers' legend groups and classes to a typical legend.

Each *LegendGroup*, in turn, contains one or more *LegendClass* objects. A *LegendClass* object represents an individual classification and has its own symbol and label—a description and format are optional. The above illustration identifies these elements in a legend.

ILegend : IMapSurround	Provides access to members that control a legend.
AutoAdd: Boolean	Indicates if a new item should be added when a new layer is added to the map.
AutoReorder: Boolean	Indicates if the legend items should be kept in the same order as the layers.
AutoVisibility: Boolean	Indicates if items should be shown only when associated layers are visible.
FlowRight: Boolean	Reserved for future use.
Format: ILegendFormat	The formatting options for the legend (can be stored in the style gallery).
Item (in Index: Long) : ILegendItem	The specified item from the legend.
ItemCount: Long	Number of items in the legend.
Title: String	Title.
AddItem (in Item: ILegendItem)	Adds a new item to the legend (to the end of the list).
ClearItems	Removes all items from the legend.
InsertItem (in Index: Long, in Item: ILegendItem)	Inserts a new item into the legend (at the location specified by index).
RemoveItem (in Index: Long)	Removes the specified item from the legend.

The *Legend*'s primary interface is *ILegend*. Use this interface to modify a legend and access its subparts.

For example, this interface provides access to the legend's items and its legend format object. *ILegend* also manages a few of the legend properties such as the title. When changing the properties of an existing legend, you must call *ILegend::Refresh* to have the changes reflected in the layout.

Formatting the legend

ILegendFormat : IUnknown	Provides access to members that control formatting information for a legend.
■ DefaultAreaPatch: IAreaPatch	Area patch. Can be overridden by the LegendItem.
■ DefaultLinePatch: ILinePatch	Line patch. Can be overridden by the LegendItem.
■ DefaultPatchHeight: Double	Patch height in points. Can be overridden by the LegendItem.
■ DefaultPatchWidth: Double	Patch width in points. Can be overridden by the LegendItem.
■ GroupGap: Double	Vertical distance in points between legend groups.
■ HeadingGap: Double	Vertical distance in points between a heading and the legend graphics that follow.
■ HorizontalItemGap: Double	Horizontal distance in points between legend item columns. Used for legends that have more than one column.
■ HorizontalPatchGap: Double	Horizontal distance in points between a patch and the legend graphics before and after.
■ LayerNameGap: Double	Vertical distance in points between layer names and the legend graphics that follow.
■ ShowTitle: Boolean	Indicates if title is visible.
■ TextGap: Double	Horizontal distance in points between labels and descriptions.
■ TitleGap: Double	Vertical distance in points between title and first legend item.
■ TitlePosition: esriRectanglePosition	Legend title position.
■ TitleSymbol: ITextSymbol	Text symbol used to draw the legend title.
■ VerticalItemGap: Double	Vertical distance in points between legend items.
■ VerticalPatchGap: Double	Vertical distance in points between patches.
← Scale (in XScale: Double, in YScale: Double)	Multiply all distances, gaps, and size property values on this interface by the specified scale factors.

The aspect of the legend can be further modified using the *ILegendFormat* interface on the *LegendFormat* object returned by *ILegend::Format*. You can also use the *IReadingDirection* interface to set the whether the legend items are aligned along the left or right side. To control the appearance of the legend more precisely you may also use a number of interface to access the individual components of the legend: the *LegendItems* and the patches. (Patches are the individual color boxes or lines associated with each legend class. For more information on patches, see the help under [IPatch](#)).

ILegendClassFormat : IUnknown	Provides access to members that control formatting information for a legend class.
■ AreaPatch: IAreaPatch	The area patch. (Optional. If non-null, this overrides default area patch specified by <i>ILegend.LegendFormat</i> .)
■ DescriptionSymbol: ITextSymbol	Text symbol used to draw legend group descriptions.
■ LabelSymbol: ITextSymbol	Text symbol used to draw the legend group labels.
■ LinePatch: ILinePatch	The line patch. (Optional. If non-null, this overrides default line patch specified by <i>ILegend.LegendFormat</i> .)
■ PatchHeight: Double	Height of the patch in points.
■ PatchWidth: Double	Width of the patch in points.

ILegendClassFormat is an interface very similar to *ILegendFormat* but at the *LegendItem* level. If the *LegendClassFormat* is not set by the renderer creating the legend the properties used to draw the *LegendItem*'s content will be the default defined in the *LegendFormat* object.

Legend items

The construction of the legend is mostly the work of the map layers through the associated *LegendItem* objects. The *Legend* can be seen as a collection of layers each layer being represented by a *LegendItem*. When the legend is refreshed, the *LegendItem* creates a set of graphic elements to display itself, pulling the information from its associated layer and the format from the objects described in the previous section. The *Legend* simply positions the title and legend item graphics relative to one another.

ILegendItem : IUnknown	Provides access to members that control how a layer appears in a legend. Can be stored in a style.
CanDisplay (in Layer: ILayer) : Boolean	Indicates if the style is compatible with the specified layer.
Columns: Integer	Number of columns in the legend item.
Graphics: IEnumElement	List of graphics that represent the legend item. Must call CreateGraphics first.
GroupIndex: Long	Zero-based index of the legend group shown by this item. Use -1 to show all legend groups using this item.
HeadingSymbol: ITextSymbol	Text symbol used to draw the heading.
Height: Double	Height of the item in points. Must call CreateGraphics first.
KeepTogether: Boolean	Indicates if classes must appear in a single column or whether they can be split across multiple columns.
Layer: ILayer	Associated layer.
LayerNameSymbol: ITextSymbol	Text symbol used to draw the layer name.
LegendClassFormat: ILegendClassFormat	Default formatting information for the legend classes. Renderer may override.
Name: String	Name of the style.
NewColumn: Boolean	Indicates if the item starts a new column in the legend.
ShowDescriptions: Boolean	Indicates if descriptions are visible.
ShowHeading: Boolean	Indicates if heading is visible.
ShowLabels: Boolean	Indicates if labels are visible.
ShowLayerName: Boolean	Indicates if layer name is visible.
Width: Double	Width of the item in points. Must call CreateGraphics first.
CreateGraphics (in Display: IDisplay, in LegendFormat: ILegendFormat)	Rebuilds the list of graphics. Call whenever the associated layer changes.

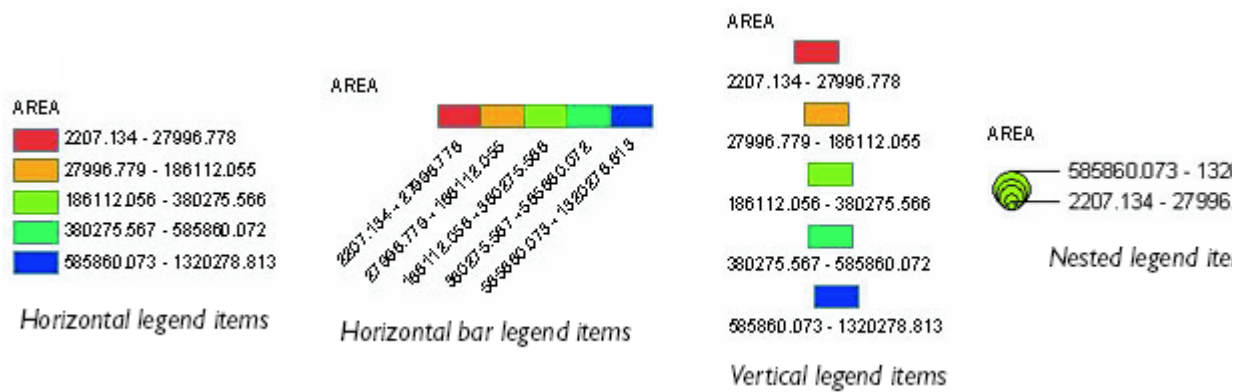
All legend items implement the *ILegendItem* interface. The interface controls all of the properties a legend item has—the layer it is associated with; the number of columns it should span; whether it should be displayed in a new column; and whether the label, description, heading, and layer name should be displayed. This interface also provides access to the legend items *LegendClassFormat* object.

Enumeration esriLegendItemArrangement	Legend item arrangement options for the order of patches, labels, and descriptions.
0 - esriPatchLabelDescription	Patch followed by label followed by description.
1 - esriPatchDescriptionLabel	Patch followed by description followed by label.
2 - esriLabelPatchDescription	Label followed by patch followed by description.
3 - esriLabelDescriptionPatch	Label followed by description followed by patch.
4 - esriDescriptionPatchLabel	Description followed by patch followed by label.
5 - esriDescriptionLabelPatch	Description followed by label followed by patch.

Horizontal and vertical legend items use the *esriLegendItemArrangement* enumeration which can be set with *IHorizontalLegendItem::Arrangement* and *IVerticalLegendItem::Arrangement* to specify the position of the label, patch, and description. The default is *esriPatchLabelDescription*, which translates to the patch on the far left, label to the right of the patch, then the description, if available, on the far right. The following illustration shows these parts of the legend in this order with a custom area patch.

	California	The Golden State
	Oregon	The Beaver State
	Washington	The Evergreen State
Patch	Label	Description

There are currently four types of legend items: *HorizontalLegendItem*, *VerticalLegendItem*, *HorizontalBarLegendItem*, and *NestedLegendItem*.



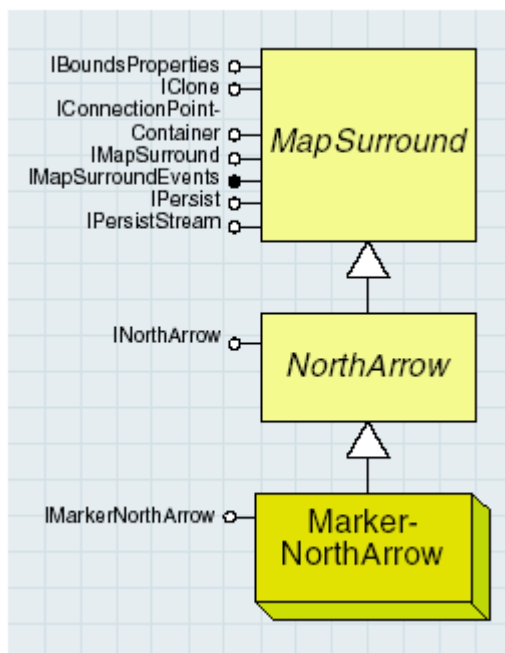
Horizontal legend items are the default and most commonly used class of legend items.

The *IHorizontalBarLegendItem* interface supports additional properties for controlling the angle of the labels above and below the patch. The default is to display the labels at a 45-degree angle.

Vertical legend items have the patches on top of the legend item text.

Nested legend items only work with graduated symbols. The image to the left shows a legend with a default nested legend item. The *INestedLegendItem* interface controls the many properties a nested legend item has, including whether or not to label the ends, the leader symbol, and the outline symbol.

North arrows



MarkerNorthArrows are character marker symbols typically coming from the ESRI North font. However, any character from any font can be used as a North arrow. *MarkerNorthArrows* implement two additional interfaces: *INorthArrow* and *IMarkerNorthArrow*.

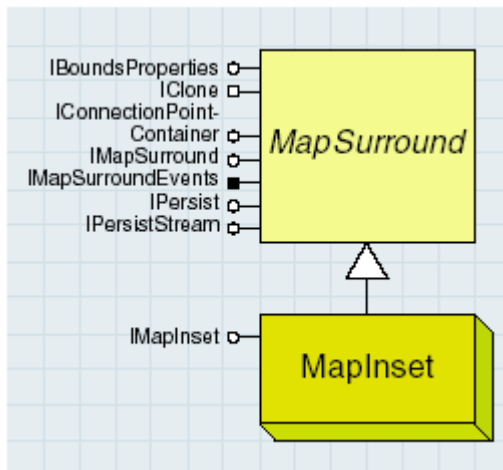
INorthArrow : IMapSurround	Provides access to members that control the north arrow.
■ Angle: Double	The counter-clockwise rotation of the north arrow in degrees. This value is calculated from the map.
■ CalibrationAngle: Double	Calibration angle. Rotation is modified by this angle.
■ Color: IColor	Color used to draw the north arrow.
■ ReferenceLocation: IPoint	The point on the map where north is calculated.
■ Size: Double	Size of the north arrow in points (1/72 inch).

The *INorthArrow* interface provides a common interface for North arrow properties, such as size, color, and reference location.

IMarkerNorthArrow : IUnknown	Provides access to members that control the Marker north arrow.
<ul style="list-style-type: none"> MarkerSymbol: IMarkerSymbol 	Symbol used to draw the north arrow. Use set to specify a marker for custom north arrows.

IMarkerNorthArrow has one property, *MarkerSymbol*, that controls which marker symbol the North arrow uses. By default, the marker symbol belongs to the ESRI North font.

Map insets

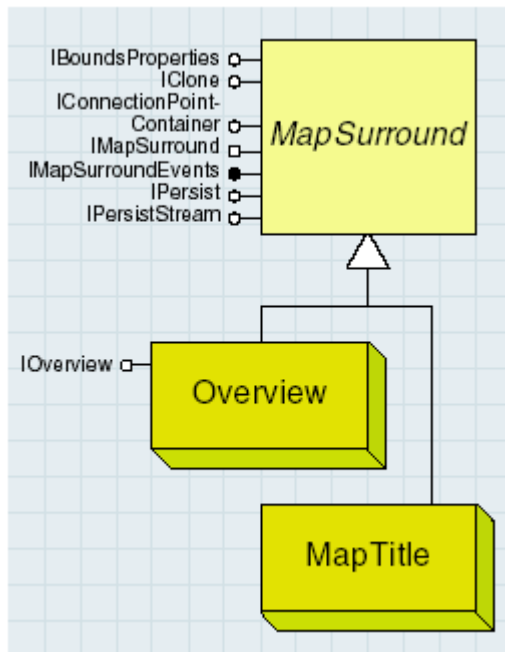


A map inset is a miniature map that typically shows a magnified view of an actual map. A *MapInset* map surround is another view of the current map extent. If you pan or zoom in on the map the *MapInset* is related to, the *MapInset* will mimic the change.

IMapInset : IMapSurround	Provides access to members that control the inset map surrounds.
<ul style="list-style-type: none"> Description: String IsLive: Boolean 	Description reflecting the current settings of the MapInset. Indicates if the inset shows a live view of the underlying map. False means a snapshot of the underlying map is taken at the time the flag is changed.
<ul style="list-style-type: none"> MapBounds: IEnvelope 	The relative position of the inset to the associated map (used when the inset is live). The zoom amount is applied to this rectangle to determine the visible bounds that is actually drawn.
<ul style="list-style-type: none"> UsingZoomScale: Boolean 	Indicates if ZoomScale or ZoomPercent is being used. The one specified last is being used.
<ul style="list-style-type: none"> VisibleBounds: IEnvelope 	The map extent shown by the inset (used when the inset is not live).
<ul style="list-style-type: none"> ZoomPercent: Double 	Zoom amount as a percentage. 100 means show the underlying map at normal size.
<ul style="list-style-type: none"> ZoomScale: Double 	The zoom amount as an absolute Scale (i.e., 1:20000).
<ul style="list-style-type: none"> CalculateVisibleBounds 	Calculates the visible bounds by applying the zoom or scale parameter to MapBounds (used when snapshot is false).

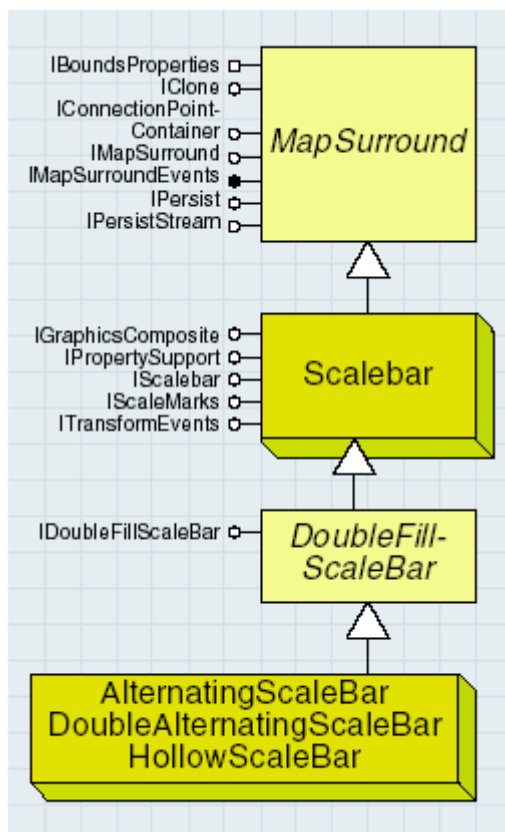
An overview map surround is the surround found in overview data windows.

Map title and overviews



The map title object is a map surround that holds a piece of text you can use to label a map. This may not be the title of the whole layout, but rather a subtitle for a specific map in the layout.

Scale bars



There are many types of scale bar map surrounds, including several types of scale lines, single-fill scale bars, and double-fill scale bars. All scale bars implement *IScaleBar* and *IScaleMarks*.

IScaleBar : IMapSurround	Provides access to members that control the scalebar map surrounds.
BarColor: IColor	Color used to draw the bar.
BarHeight: Double	Height of the bar in points (1/72 inch).
Division: Double	Number of units in one major division.
Divisions: Integer	Total number of divisions (including those before zero).
DivisionsBeforeZero: Integer	Number of divisions to the left of zero.
LabelFrequency: tagesriScaleBarFrequency	The label style indicating which marks are labeled.
LabelGap: Double	Vertical gap between the bar and the labels in points (1/72 inch).
LabelPosition: tagesriVertPosEnum	Vertical positioning of the mark labels.
LabelSymbol: ITextSymbol	Symbol used to draw the labels.
NumberFormat: INumberFormat	Number format.
ResizeHint: tagesriScaleBarResizeHint	Indicates what happens when scale bar is resized.
Subdivisions: Integer	Number of subdivisions per major division.
UnitLabel: String	The unit label.
UnitLabelGap: Double	Gap between the scale bar and the unit label in points (1/72 inch).
UnitLabelPosition: tagesriScaleBarPos	Vertical positioning of the unit label.
UnitLabelSymbol: ITextSymbol	Unit label symbol.
Units: esriUnits	The units reported.
UseMapSettings	Sets division and units based on map.

The *IScaleBar* interface manages most of the properties a scale bar has, including bar color, bar height, division, and label frequency.

IScaleMarks : IUnknown	Provides access to members that control the scale bar mark properties.
DivisionMarkHeight: Double	Height of division marks in points (1/72 inch). Use <i>esriAutoScaleBar</i> to automatically calculate.
DivisionMarkSymbol: ILineSymbol	Symbol used to draw the division marks.
MarkFrequency: tagesriScaleBarFrequency	Mark frequency.
MarkPosition: tagesriVertPosEnum	Vertical positioning of the marks relative to the bar.
SubdivisionMarkHeight: Double	Height of subdivision marks in points (1/72 inch). Use <i>esriAutoScaleBar</i> to automatically calculate.
SubdivisionMarkSymbol: ILineSymbol	Symbol used to draw the subdivision marks.

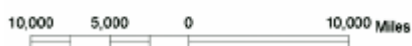
The *IScaleMarks* interface manages all of the properties of a scale bar that relate to the individual marks, including the division and subdivision marks heights and symbols, the marks frequency, and their position.



Alternating scale bar



Double alternating scale bar



Hollow scale bar

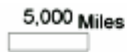
Double-fill scale bars are the most advanced scale bars. These use two symbols to create an attractive scale bar. There are currently three types of double-fill scale bars: alternating, double-alternating, and hollow. The graphic above shows an example of each.

IDoubleFillScaleBar : IUnknown
<div> <div></div> <div>FillSymbol1: IFillSymbol</div> </div> <div> <div></div> <div>FillSymbol2: IFillSymbol</div> </div>

Provides access to members that control a scale bar that uses two fill symbols to draw bar.

Symbol used to draw the bar.
Symbol used to draw the bar.

All double-fill scale bars implement the *IDoubleFillScaleBar* interface. This interface manages the two fill symbols used when rendering the scale bar.



Single division scale bar

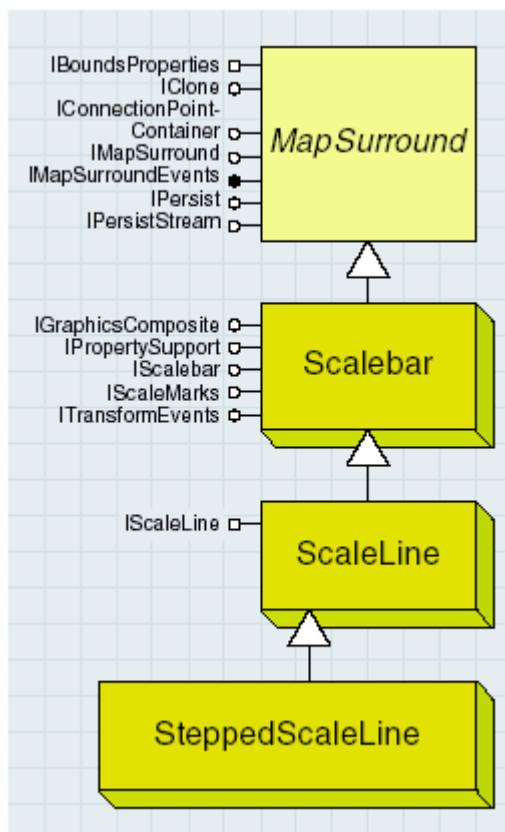
Single-fill scale bars are similar to double-fill scale bars except they use one fill symbol. ArcMap currently has one single-fill scale bar, the *SingleDivisionScaleBar*. The graphic above shows an example of a single-division scale bar.

ISingleFillScaleBar : IUnknown
<div> <div></div> <div>FillSymbol: IFillSymbol</div> </div>

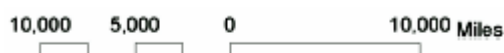
Provides access to members that control a scale bar that uses a single fill symbol to draw bar.

Symbol used to draw the bar.

The *ISingleFillScaleBar* interface manages the single-fill symbol used by scale bars of this type.



Scale line scale bars are the only class of scale bars that represent a scale bar as a line. ArcMap currently has one type of scale line scale bar—the stepped-line scale bar.

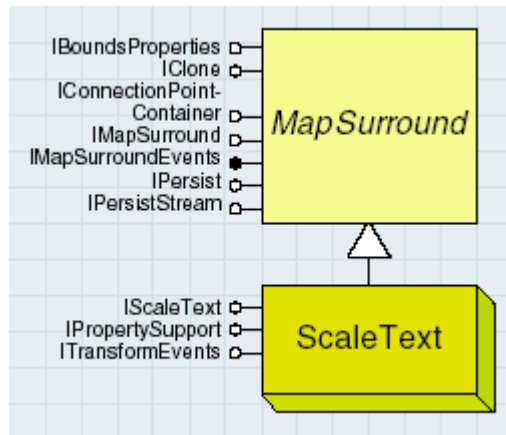


Stepped-line scale bar

Scale lines are another class of scale bars that are based on line work instead of polygons. The graphic above shows an example of a stepped-line scale bar.

IScaleLine : IUnknown	Provides access to members that control a line scale bar.
<div> <div>LineSymbol: ILineSymbol</div> </div>	<div>Symbol used to draw the line.</div>

The *IScaleLine* interface manages the one line symbol used by scale lines.



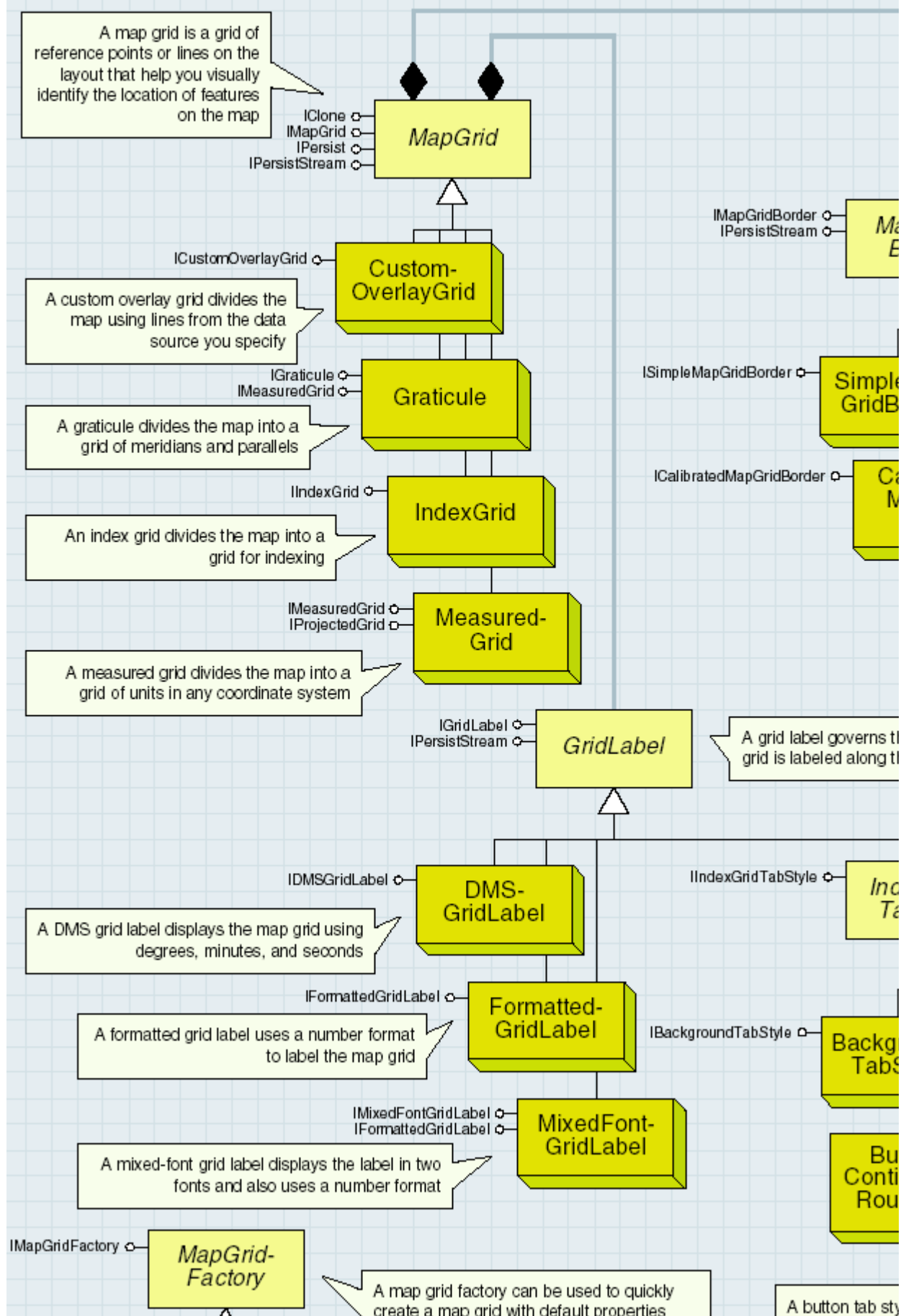
ScaleText is essentially a text element that describes the map's scale. One example of scale text is "1 inch equals 2,400 miles".

IScaleText : IMapSurround	Provides access to members that control the scale text.
<div> <div>Format: String</div> <div>MapUnitLabel: String</div> <div>MapUnits: esriUnits</div> <div>NumberFormat: INumberFormat</div> <div>PageUnitLabel: String</div> <div>PageUnits: esriUnits</div> <div>Style: tagesriScaleTextStyleEnum</div> <div>Symbol: ITextSymbol</div> <div>Text: String</div> </div>	<div>Format of the scale text. Style must be set to custom.</div> <div>Map unit label of the scale text. Style must be set to relative.</div> <div>Map units of the scale text. Style must be set to custom.</div> <div>Number formatting.</div> <div>Page unit label of the scale text. Style must be set to relative.</div> <div>Page units of the scale text. Style must be set to custom.</div> <div>Style of the scale text.</div> <div>Symbol of the scale text.</div> <div>The scale text.</div>

The interface *IScaleText* controls the format of the string that is added as a map surround element. This interface has properties such as *MapUnits* and *MapUnitLabel*, *PageUnits* and *PageUnitLabel*, and *Text*, which combines the label properties into a sentence.

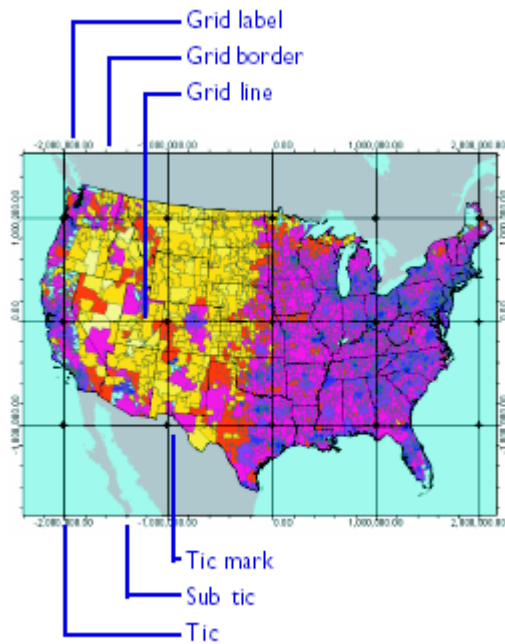
Map grids

ArcMap map grid obj



The Carto library contains a rich set of map grid objects for displaying index and geographic reference grids on a map. A map grid can be a grid of geographic or projected coordinates, or a reference grid like those found in street maps. Map grids are part of the layout of a map and can only be seen in layout view.

The map grid factories are described in the [CartoUI Library Overview](#).



Parts of a map grid

To get to a map grid programmatically, navigate to the *PageLayout* coclass, then use its *IGraphicsContainer* interface's *FindFrame* method to get to the *Map's MapFrame*. The *MapFrame* coclass has an *IMapGrids* interface from which you can get to all the map grids for that dataframe.

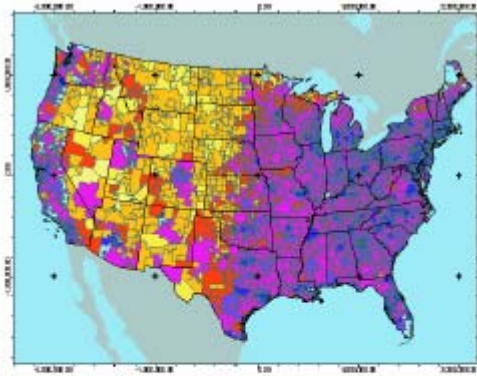
[Visual Basic 6.0]

```
Public Sub FindMapGrid()
    Dim pMap As IMap, pMxDoc As IMxDocument
    Dim pMapFrame As IMapFrame
    Dim pGraphicsContainer As IGraphicsContainer
    Dim pMapGrid As IMapGrid
    Set pMxDoc = ThisDocument
    Set pMap = pMxDoc.FocusMap
    Set pGraphicsContainer = pMxDoc.PageLayout
    Set pMapFrame = pGraphicsContainer.FindFrame(pMap)
    Dim pMapGrids As IMapGrids
    Set pMapGrids = pMapFrame
    If pMapGrids.MapGridCount > 0 Then
        Set pMapGrid = pMapGrids.MapGrid(0)
    Else
        MsgBox "No grid found."
    End If
End Sub
```

IMapGrid : IUnknown	Provides access to members that control a map grid.
<ul style="list-style-type: none"> ■ Border: IMapGridBorder ■ ExteriorWidth (in pDisplay: IDisplay, in pMapFrame: IMapFrame) : Double ■ LabelFormat: IGridLabel ■ LineSymbol: ILineSymbol ■ Name: String ■ SubTickCount: Integer ■ SubTickLength: Double ■ SubTickLineSymbol: ILineSymbol ■ TickLength: Double ■ TickLineSymbol: ILineSymbol ■ TickMarkSymbol: IMarkerSymbol ■ Visible: Boolean 	<p><i>The map grid border.</i></p> <p><i>The width (in display units) of the portion of the grid that is outside of the frame.</i></p> <p><i>The label format for map grid labels.</i></p> <p><i>The symbol used to draw grid lines - null will draw no lines.</i></p> <p><i>The name of the map grid.</i></p> <p><i>The number of subticks to draw between the major ticks.</i></p> <p><i>The length of the subticks in points.</i></p> <p><i>The symbol used to draw the subtick lines.</i></p> <p><i>The length of the major ticks in points.</i></p> <p><i>The line symbol used to draw the major ticks.</i></p> <p><i>The symbol used to draw tick marks at the grid interval intersections - null will draw no tick marks.</i></p> <p><i>Indicates if the map grid is visible.</i></p>
<ul style="list-style-type: none"> ← Draw (in Display: IDisplay, in pMapFrame: IMapFrame) ← GenerateGraphics (in pMapFrame: IMapFrame, in GraphicsContainer: IGraphicsContainer) ← PrepareForOutput (in hdc: Long, in dpi: Long, in PixelBounds: tagRECT, in pMapFrame: IMapFrame) ← QueryLabelVisibility (out leftVis: Boolean, out topVis: Boolean, out rightVis: Boolean, out bottomVis: Boolean) ← QuerySubTickVisibility (out leftVis: Boolean, out topVis: Boolean, out rightVis: Boolean, out bottomVis: Boolean) ← QueryTickVisibility (out leftVis: Boolean, out topVis: Boolean, out rightVis: Boolean, out bottomVis: Boolean) ← SetDefaults (in pMapFrame: IMapFrame) ← SetLabelVisibility (in leftVis: Boolean, in topVis: Boolean, in rightVis: Boolean, in bottomVis: Boolean) ← SetSubTickVisibility (in leftVis: Boolean, in topVis: Boolean, in rightVis: Boolean, in bottomVis: Boolean) ← SetTickVisibility (in leftVis: Boolean, in topVis: Boolean, in rightVis: Boolean, in bottomVis: Boolean) 	<p><i>Draws the map grid for a map frame to the given display.</i></p> <p><i>Generates graphic elements corresponding to the grid lines and stores them in the specified graphics container.</i></p> <p><i>Prepares the map grid for output to a device.</i></p> <p><i>Returns the visibility of the labels along all four sides of the map grid.</i></p> <p><i>Returns the visibility of the subticks along all four sides of the map grid.</i></p> <p><i>Returns the visibility of the ticks along all four sides of the map grid.</i></p> <p><i>Sets the properties of the map grid to default values.</i></p> <p><i>Sets the visibility of the labels along all four sides of the map grid.</i></p> <p><i>Sets the visibility of the subticks along all four sides of the map grid.</i></p> <p><i>Sets the visibility of the ticks along all four sides of the map grid.</i></p>

IMapGrid provides access to the methods and properties common to all types of map grids. The *Draw* method can be used to draw a map grid to, for example, a PictureBox control that has a map and display associated with it. The *PrepareForOutput* method takes a device's HDC and should be called before the *Draw* method.

When you create a new map grid, you have to populate the properties of the grid that *IMapGrid* exposes. The following code illustrates how you can do this. After doing this, you can populate the properties exposed by interfaces specific to the grid type, then add the grid to a data frame.



If you want tick marks in your grid, you can create a marker symbol and assign it to the IMapGrid::TickMarkSymbol property. If you do not want either a TickMarkSymbol or a TickLineSymbol, set these properties to 'Nothing'.

The following example shows how to create a custom grid by code. Modify its properties and labeling and add it the map frame. It is best to use cartographic line symbol so that the grids lines have square butts.

[Visual Basic 6.0]

```
Public Sub CreateGrid()

'Create the grid
Dim pMapGrid As IMapGrid
Set pMapGrid = New Graticule
pMapGrid.Name = "Map Grid"

'Create a color
Dim pColor As IColor
Set pColor = New RGBColor
pColor.RGB = &HBBBBBB ' -> Gray

'Set the line symbol used to draw the grid
Dim pLineSymbol As ICartographicLineSymbol
Set pLineSymbol = New CartographicLineSymbol
pLineSymbol.Cap = esriLCSButt
pLineSymbol.Width = 2
pLineSymbol.Color = pColor
pMapGrid.LineSymbol = pLineSymbol

pMapGrid.Border = Nothing ' clear the default frame border

'Set the Tick Properties
pMapGrid.TickLength = 15
Set pLineSymbol = New CartographicLineSymbol
pLineSymbol.Cap = esriLCSButt
pLineSymbol.Width = 1
pLineSymbol.Color = pColor
pMapGrid.TickLineSymbol = pLineSymbol
pMapGrid.TickMarkSymbol = Nothing

'Set the Sub Tick Properties
pMapGrid.SubTickCount = 5
pMapGrid.SubTickLength = 10
Set pLineSymbol = New CartographicLineSymbol
pLineSymbol.Cap = esriLCSButt
pLineSymbol.Width = 0.2
pLineSymbol.Color = pColor
pMapGrid.SubTickLineSymbol = pLineSymbol

' Set the Grid labels properties
Dim pGridLabel As IGridLabel
Set pGridLabel = pMapGrid.LabelFormat
pGridLabel.LabelOffset = 15

'Set the Tick, SubTick, Label Visibility along the 4 sides of the grid
pMapGrid.SetTickVisibility True, True, True, True
```

```

pMapGrid.SetSubTickVisibility True, True, True, True
pMapGrid.SetLabelVisibility True, True, True, True

'Make map grid visible, so it gets drawn when Active View is updated
pMapGrid.Visible = True







'Set the IMeasuredGrid properties
Dim pMeasuredGrid As IMeasuredGrid
Set pMeasuredGrid = pMapGrid
pMeasuredGrid.FixedOrigin = True
pMeasuredGrid.XIntervalSize = 10 'meridian interval
pMeasuredGrid.XOrigin = 5 'shift grid 5°
pMeasuredGrid.YIntervalSize = 10 'parallel interval
pMeasuredGrid.YOrigin = 5 'shift grid 5°

' Add the grid to the MapFrame
Dim pMap As IMap
Dim pMxDoc As IMxDocument
Dim pGraphicsContainer As IGraphicsContainer
Dim pMapFrame As IMapFrame
Set pMxDoc = ThisDocument
Set pMap = pMxDoc.FocusMap
Set pGraphicsContainer = pMxDoc.PageLayout
Set pMapFrame = pGraphicsContainer.FindFrame(pMap)
Dim pMapGrids As IMapGrids
Set pMapGrids = pMapFrame
pMapGrids.AddMapGrid pMapGrid

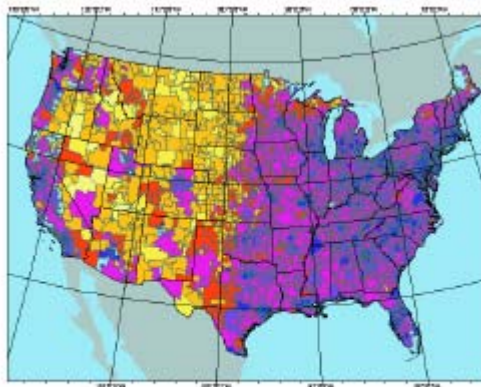
'Refresh the view
Dim pActiveView As IActiveView
Set pActiveView = pMxDoc.PageLayout
pActiveView.PartialRefresh esriViewBackground, Nothing, Nothing

End Sub

```

IMeasuredGrid : IUnknown	Provides access to the members that control the lines that make up the map grid.
 FixedOrigin: Boolean	Indicates if the origin is read from the XOrigin and YOrigin properties (true) or if it is computed dynamically from the data frame (false).
 Units: esriUnits	The units for the intervals and origin.
 XIntervalSize: Double	The interval between grid lines along the X axis.
 XOrigin: Double	The origin of the grid on the X axis.
 YIntervalSize: Double	The interval between grid lines along the Y axis.
 YOrigin: Double	The origin of the grid on the Y axis.






The *IMeasuredGrid* interface is implemented by the *MeasuredGrid* and *Graticule* coclasses. It exposes information on the origins, intervals, and units of the grid. If you set *IMeasuredGrid::FixedOrigin* to False, the origin is computed from the data frame instead of from the x- and y-origin properties. *IMeasuredGrid::Units* need not be populated for a graticule.



A graticule divides the map by meridians and parallels.

Index grids

An index grid is a map grid that divides the map into the specified number of columns and rows. It is mainly used to index a map.

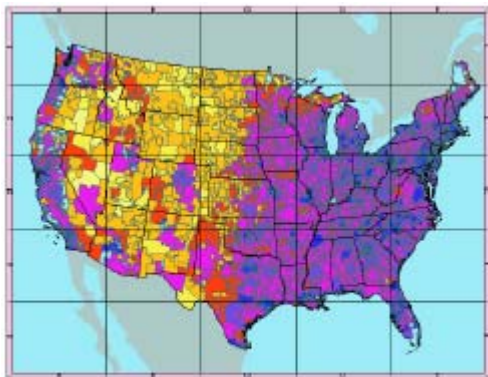
IIndexGrid : IMapGrid	Provides access to members that control the index grid.
<div> <div>  ColumnCount: Long </div> <div>  RowCount: Long </div> <div>  XLabel (in column: Long) : String </div> <div>  YLabel (in Row: Long) : String </div> </div>	<div> <div>The number of columns in the index grid.</div> <div>The number of rows in the index grid.</div> <div>The label for the given column in the index grid.</div> <div>The label for the given row in the index grid.</div> </div>
<div>  QueryCellExtent (in Row: Long, in column: Long, in pMapFrame: IMapFrame, Extent: IEnvelope) </div>	<div>Provides access to the cell extent in page space for the given row and column.</div>

IIndexGrid gives you access to the functionality common to all index grids. Using the *XLabel* and the *YLabel* properties, you can set or retrieve the label for each column and index in the grid. You can create an index grid as illustrated in the sample below:

[Visual Basic 6.0]

```
'Create indexgrid
Dim pMapGrid As IMapGrid
Dim pIndexGrid As IIndexGrid
Set pIndexGrid = New IndexGrid
Set pMapGrid = pIndexGrid
'Set the IIndexGrid properties
pIndexGrid.ColumnCount = 5
pIndexGrid.RowCount = 5
'Set grid label strings for the x and y axes
Dim i As Integer
For i = 0 To (pIndexGrid.ColumnCount - 1)
    pIndexGrid.XLabel(i) = VBA.Chr(i + Asc("A"))
Next i
For i = 0 To (pIndexGrid.RowCount - 1)
    pIndexGrid.YLabel(i) = VBA.Str(i + 1)
Next i
```


IIndexGrid::QueryCellExtent is useful for finding the features that cross a cell in the grid. You can use the envelope returned by this method in a spatial filter after transforming it into map coordinates using *IDisplayTransformation::TransformRect*. You can use this filter to search for the features that cross this cell in the grid and to create an index listing of features and their location on the grid.



An index grid

Measured grid

A measured grid is a map grid with grid lines on a coordinate system specified using the *IProjectedGrid* interface.

IProjectedGrid : IUnknown	Provides access to members that control the projection information for map grids.
<div>  SpatialReference: ISpatialReference </div>	<div>The spatial reference system of the grid.</div>

The *IProjectedGrid* interface holds the spatial reference information associated with a measured grid. If you want to create a measured grid in the same projection as the data frame it is in, you can set the *IProjectedGrid::SpatialReference* property using the data frame's *IMap::SpatialReference* property.

A measured grid divides the map into a grid of units in a coordinate system of your choice. The grid can be in a projected coordinate system or in a geographic coordinate system. A measured grid in a geographic coordinate system is equivalent to a graticule. A measured grid can be in the same spatial reference system as the data frame or in a different one. To create a measured grid with a different projection, you should first create an instance of a coclass that inherits from *SpatialReference*. You can then set the *IProjectedGrid::SpatialReference* property of the grid with the *ISpatialReference* interface of this object. The following example shows how to create a measured grid and set the properties exposed through its specific interfaces.

[Visual Basic 6.0]

```
'Create measuredgrid
Dim pMapGrid As IMapGrid, pMeasuredGrid As IMeasuredGrid
Set pMeasuredGrid = New MeasuredGrid
Set pMapGrid = pMeasuredGrid
'Set the IMeasuredGrid properties
'Origin coordinates and interval sizes are in map units
pMeasuredGrid.FixedOrigin = True
pMeasuredGrid.Units = m_pMap.MapUnits
pMeasuredGrid.XIntervalSize = 1000000 'meridian interval
pMeasuredGrid.XOrigin = -3000000
pMeasuredGrid.YIntervalSize = 1000000 'parallel interval
pMeasuredGrid.YOrigin = -3000000
'Set the IProjectedGrid properties
Dim pProjectedGrid As IProjectedGrid
Set pProjectedGrid = pMeasuredGrid
Set pProjectedGrid.SpatialReference = m_pMap.SpatialReference
```

Custom overlay grids

A custom overlay grid is a map grid with grid lines read from a feature.

ICustomOverlayGrid : IMapGrid	Custom Overlay Grid Interface
<ul style="list-style-type: none"> DataSource: IFeatureClass LabelField: String 	<p>Sets or returns the data source containing the grid cells</p> <p>Sets or returns the name of the field used to label the grid</p>

The *ICustomOverlayGrid* interface gives you access to the feature class that the grid lines are read from through the *ICustomOverlayGrid::DataSource* property. It also lets you specify which field in this feature class will label the grid using the *ICustomOverlayGrid::LabelField* property.

Map grid borders

The map grid border coclasses determine how the outline of a map grid is drawn. Using the *IMapGridBorder* interface, you can find the width of the map grid border. Using the *DisplayName* property, you can report the type of the border object to which the *IMapGridBorder* interface is pointing. The table below lists the strings reported by this property for the two border types.

Type of MapGridBorder	Display Name
SimpleMapGridBorder	"Simple Border"
CalibratedMapGridBorder	"Calibrated Border"

IMapGridBorder : IUnknown	Provides access to members that control the map grid border.
<ul style="list-style-type: none"> DisplayName: String Width: Double 	<p>The display name for the map grid border.</p> <p>The width of the map grid border in points.</p>
<ul style="list-style-type: none"> Draw (in Display: IDisplay, in frameGeometry: IGeometry, in mapGeometry: IGeometry) 	<p>Draws the border to the specified display, using the frame bounds and the map bounds in page space.</p>

When you create a new map grid border, you don't need to use the *IMapGridBorder* interface. As you can see, all the properties exposed by this interface are read-only.

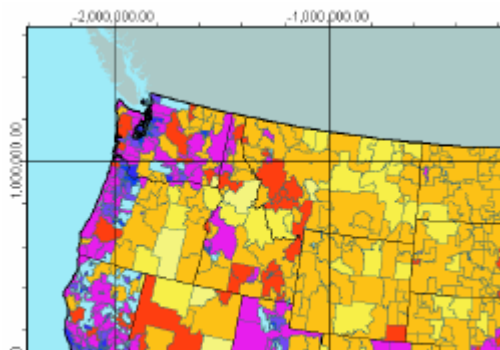
A simple map grid border is drawn using a line symbol specified with the *ISimpleMapGridBorder* interface.

ISimpleMapGridBorder : IUnknown	Provides access to the members that control the simple map grid border.
<div> <div> <div></div> <div>LineSymbol: ILineSymbol</div> </div> </div>	<i>The line symbol used to draw the border.</i>

The *ISimpleMapGridBorder* interface provides access to the line symbol used to draw the grid border through the *LineSymbol* property. The code below illustrates how you can create a simple map grid border.

[Visual Basic 6.0]

```
'Create a simple map grid border
Dim pSimpleMapGridBorder As ISimpleMapGridBorder
Set pSimpleMapGridBorder = New SimpleMapGridBorder
'Set the ISimpleMapGridBorder properties
Dim pLineSymbol As ISimpleLineSymbol
Set pLineSymbol = New SimpleLineSymbol
pLineSymbol.Style = esriSLSSolid
pLineSymbol.Color = BuildRGB(0, 0, 0)
pLineSymbol.Width = 2
pSimpleMapGridBorder.LineSymbol = pLineSymbol
'Assign this border to the map grid
pMapGrid.Border = pSimpleMapGridBorder
```



A simple map grid border

Calibrated map grid borders

The *CalibratedMapGridBorder* coclass encapsulates the functionality required to draw a map grid outline composed of a graduated band.

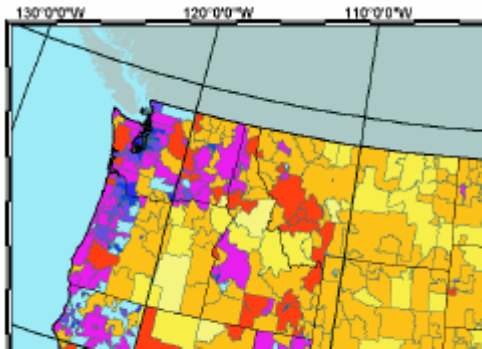
ICalibratedMapGridBorder : IUnknown	Provides access to members that control the calibrated map grid border.
<div> <div> <div></div> <div>Alternating: Boolean</div> </div> </div>	<i>Indicates if the border pattern alternates across the width of the border.</i>
<div> <div> <div></div> <div>BackgroundColor: IColor</div> </div> </div>	<i>The background color of the border pattern.</i>
<div> <div> <div></div> <div>BorderWidth: Double</div> </div> </div>	<i>The width of the border in points.</i>
<div> <div> <div></div> <div>ForegroundColor: IColor</div> </div> </div>	<i>The foreground color of the border pattern.</i>
<div> <div> <div></div> <div>Interval: Double</div> </div> </div>	<i>The interval between border patterns in points.</i>

You can use the *ICalibratedMapGridBorder* interface to set or retrieve the properties of a calibrated map grid border, such as the foreground and background color of the pattern, the interval of the pattern, the background color of the band, and the width of the border. If you want the pattern to alternate in two bands across the width of the border, set the *Alternating* property to True. Setting this property to False will produce a border with a single band of the pattern.

[Visual Basic 6.0]

```
'Create a calibrated map grid border
Dim pCalibratedBorder As ICalibratedMapGridBorder
Set pCalibratedBorder = New CalibratedMapGridBorder
'Set ICalibratedMapGridBorder properties
pCalibratedBorder.BackgroundColor = BuildRGB(255, 255, 255)
pCalibratedBorder.ForegroundColor = BuildRGB(0, 0, 0)
pCalibratedBorder.BorderWidth = 10
pCalibratedBorder.Interval = 72
pCalibratedBorder.Alternating = True 'Double alternating border
'Assign this border to the map grid
pMapGrid.Border = pCalibratedBorder
```

The interval of the pattern on the band is in points and page units. If you want to compute your border intervals in map units, you can use a *DisplayTransformation* to convert your interval from map units to page units. You can convert these to points, considering that there are 72 points to an inch.



A calibrated map grid border

Grid labels

A grid label object is associated with every map grid object and provides the functionality required to draw labels around the map grid.

IGridLabel : IUnknown	
<ul style="list-style-type: none"> Applies (in grid: IMapGrid) : Boolean Color: IColor DisplayName: String EditObject: IUnknown Pointer Font: Font LabelAlignment (in axis: esriGridAxisEnum) : Boolean LabelOffset: Double 	<p>Provides access to members that control the way a map grid is labeled.</p> <p>Indicates if this grid label can be used with the specified map grid.</p> <p>The color of the grid label.</p> <p>The display name for the type of grid label.</p> <p>The interface to an object that can be edited with a property sheet.</p> <p>The object is either the grid label itself or a single editable property.</p> <p>The font used by the grid label.</p> <p>Indicates if the grid label is horizontal (true) or vertical (false) on the specified axis.</p> <p>The offset of the grid label from the border in points.</p>
<ul style="list-style-type: none"> Draw (in labelValue: Double, in Location: IPoint, in axis: esriGridAxisEnum, in Display: IDisplay) 	<p>Draws a label on the specified grid axis.</p>
<ul style="list-style-type: none"> Preview (in hdc: Long, in rectangle: tagRECT) 	<p>Draws a preview of the grid label into the specified hdc.</p>
<ul style="list-style-type: none"> QueryTextExtent (in labelValue: Double, in Location: IPoint, in axis: esriGridAxisEnum, in Display: IDisplay, Extent: IEnvelope) 	<p>Determines the extent of a label's text on the specified grid axis.</p>

The *IGridLabel* interface holds properties common to all types of grid labels. Not all grid labels can be used with all types of grids. The *Applies* property of *IGridLabel* returns True if the grid label can be used with the grid that you pass in as argument. The table below lists the types of labels that can be used with each grid type.

Using the *IGridLabel::DisplayName* property, you can list the type of label that the *IGridLabel* interface is pointing to. The strings returned for the various label types are also listed in the table below.

Grid	Label	DisplayName
Graticule	DMSLabel	Degrees Minutes Seconds
Measured-Grid	FormattedLabel	Formatted
	MixedFontLabel	Mixed Font
Index-Grid	ButtonTabStyle	Button Tabs
	RoundedTabStyle	Rounded Tabs
	ContinuousTabStyle	Continuous Tabs
	BackgroundTabStyle	Filled Background

You can control the vertical or horizontal orientation of the labels along each of the four sides of the grid using the *IGridLabel::LabelAlignment* property. You specify which axis you are setting the property for using an *esriGridAxisEnum* enumeration.

Enumeration esriGridAxisEnum

0 - esriGridAxisNone
 1 - esriGridAxisTop
 2 - esriGridAxisBottom
 3 - esriGridAxisLeft
 4 - esriGridAxisRight

Map grid axes.

No axis.
 Top axis.
 Bottom axis.
 Left axis.
 Right axis.

Here's how you would populate the properties exposed by *IGridLabel* for a newly created *GridLabel*:

[Visual Basic 6.0]

```
' Create grid label
Dim pGridLabel As IGridLabel
Set pGridLabel = New DMSGridLabel
' Set font and color
Dim pFont As IFontDisp
Set pFont = New StdFont
pFont.Name = "Arial"
pFont.size = 24
pGridLabel.Font = pFont
pGridLabel.Color = BuildRGB(0, 0, 0)
'Specify Vertical Labels
pGridLabel.LabelAlignment(esriGridAxisLeft) = False
pGridLabel.LabelAlignment(esriGridAxisRight) = False
pGridLabel.LabelOffset = 6
```

You would then set the properties specific to the type of grid label you are creating. You would associate the newly created grid label to the grid using the grid's *IMapGrid::LabelFormat* property:

[Visual Basic 6.0]

```
pMapGrid.LabelFormat = pGridLabel
```

IGridLabel::QueryTextExtent is used to check for labeling conflicts by ArcMap. The *IGridLabel::EditObject* method is used in the *MapGrid* property pages. It returns an interface that determines which dialog box is brought up when a user clicks Additional Properties under the Labels tab. The interfaces returned for each of the label types are listed in the table below.

Grid label	Edit object returned
DMSGridLabel	IDMSGridLabel
FormattedGridLabel	INumberFormat
MixedFontGridLabel	IMixedFontGridLabel
IndexFontGridLabel	IIndexGridTabStyle

DMS grid labels

A DMS grid label labels the map grid using degrees, minutes, and seconds. You can use the *DMSGridLabel* coclass to label graticules.

IDMSGridLabel : IUnknown	Provides access to members that control the DMS Grid Label.
■ LabelType: esriDMSGridLabelType	The type of the DMS grid label.
■ LatLonFormat: ILatLonFormat	The format with which the latitudes and longitudes are displayed.
■ MinutesColor: IColor	The color used to display the minutes.
■ MinutesFont: Font	The font used to display the minutes.
■ SecondsColor: IColor	The color used to display the seconds.
■ SecondsFont: Font	The font used to display the seconds.
■ ShowZeroMinutes: Boolean	Indicates if zero minutes are shown.
■ ShowZeroSeconds: Boolean	Indicates if zero seconds are shown.

IDMSGridLabel provides access to the font, color, and format information required to create a DMS grid label. The *LabelType* property can be set using the *esriDMSGridLabelType* enumeration, which is listed below. Only the *esriDMSGridLabelStandard* and *esriDMSGridLabelStacked* values are currently implemented.

You can use a standard label to create a DMS label with the degrees, minutes, and seconds on the same line. A stacked label has the minutes stacked over the seconds, with both in smaller font size.

Enumeration esriDMSGridLabelType

- 0 - esriDMSGridLabelStandard
- 1 - esriDMSGridLabelStacked
- 2 - esriDMSGridLabelDD
- 3 - esriDMSGridLabelDM
- 4 - esriDMSGridLabelDS

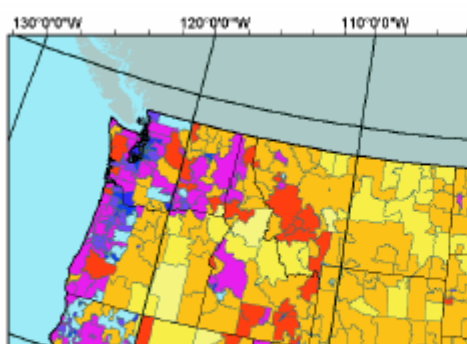
DMS grid label type options.

- Standard.
- Minutes stacked over seconds.
- Decimal degrees.
- Decimal minutes.
- Decimal seconds.

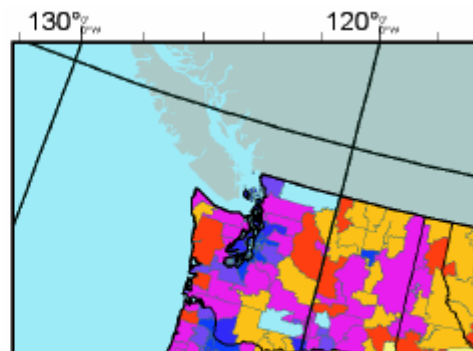
The following code demonstrates how to create a DMS grid label:

[Visual Basic 6.0]

```
'Create a DMS grid label
Dim pDMSLabel As IDMSGridLabel
Set pDMSLabel = New DMSGridLabel
'Set IDMSGridLabel properties
pDMSLabel.LabelType = esriDMSGridLabelStandard
pDMSLabel.ShowZeroMinutes = True
pDMSLabel.ShowZeroSeconds = True
Dim pLatLonFormat As ILatLonFormat
Set pLatLonFormat = New LatLonFormat
pLatLonFormat.ShowDirections = True
pDMSLabel.LatLonFormat = pLatLonFormat
Dim pFont As IFontDisp
Set pFont = New StdFont
pFont.Bold = False
pFont.Name = "Arial"
pFont.Italic = False
pFont.Underline = False
pFont.Size = 8
pDMSLabel.MinutesFont = pFont
pDMSLabel.MinutesColor = BuildRGB(0, 0, 0)
pDMSLabel.SecondsFont = pFont
pDMSLabel.SecondsColor = BuildRGB(0, 0, 0)
```



DMS grid label set to
esriDMSGridLabelStandard




DMS grid label set to
esriDMSGridLabelStacked

Formatted grid labels

The *FormattedGridLabel* coclass uses any of the coclasses that inherits from the *NumberFormat* abstract class to create the grid labels.

IFormattedGridLabel : IUnknown

 **Format:** INumberFormat

Provides access to members controlling the number format of a grid label.

The format used to display the numbers in the grid label.

This interface has a *Format* property that takes an *INumberFormat* interface. The following code illustrates the creation of a formatted grid label:

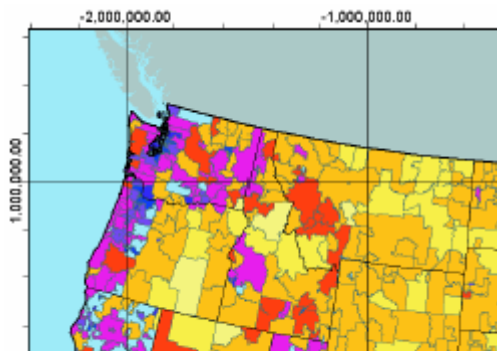
[Visual Basic 6.0]

```
'Create the label
Dim pFormattedGridLabel As IFormattedGridLabel
Set pFormattedGridLabel = New FormattedGridLabel
'Set IFormattedGridLabel properties
Dim pNumericFormat As INumericFormat
Set pNumericFormat = New NumericFormat
```

```

pNumericFormat.AlignmentOption = esriAlignRight
pNumericFormat.RoundingOption = esriRoundNumberOfDecimals
pNumericFormat.RoundingValue = 2
pNumericFormat.ShowPlusSign = False
pNumericFormat.UseSeparator = True
pNumericFormat.ZeroPad = True
pFormattedGridLabel.Format = pNumericFormat

```



A measured grid with formatted grid labels

Mixed font grid labels

A mixed font grid label uses two fonts to display the label. It also uses a number format to format the label string. Use the *MixedFontGridLabel* coclass to label map grids in two fonts and in the format specified using the *IFormattedGridLabel* interface.

IMixedFontGridLabel : IUnknown	<i>Provides access to members that define the appearance of the secondary group of digits in the grid label.</i>
■ NumGroupedDigits: Integer	<i>The number of digits that are displayed in the secondary font and color.</i>
■ SecondaryColor: IColor	<i>The color of the second group of digits.</i>
■ SecondaryFont: Font	<i>The font used for the second group of digits.</i>

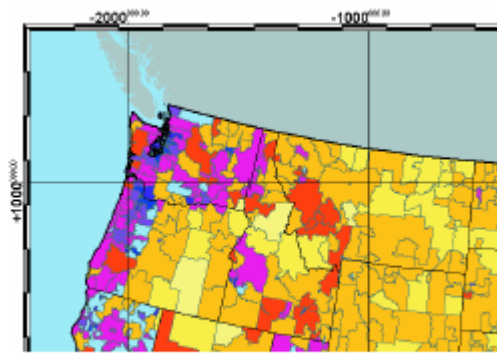
The *IMixedFontGridLabel::NumberOfDigits* property determines how the two fonts are applied to the label string. The last n digits of the label— where n is the number assigned as the *NumberOfDigits*—are displayed in the secondary font and color. The remaining digits are displayed in the primary font and color. The primary font and color are set using *IGridLabel::Font* and *IGridLabel::Color*. The secondary font and color are set using *IMixedFontGridLabel::SecondaryFont* and *IMixedFontGridLabel::SecondaryColor*. The following code illustrates how you can create a mixed font grid label:

[Visual Basic 6.0]

```

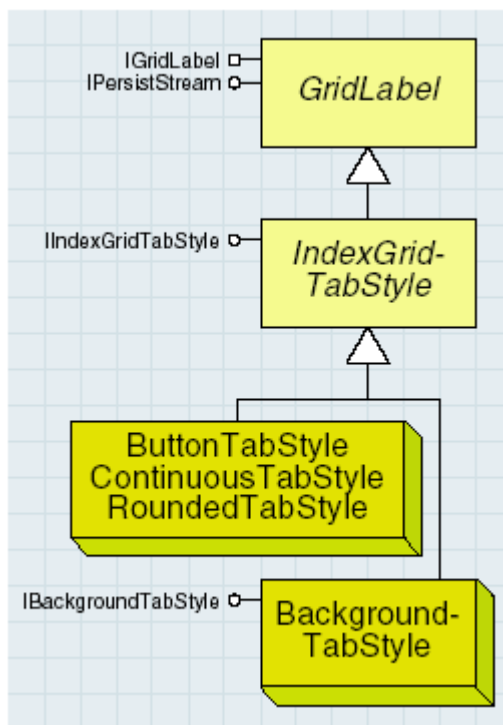
'Create the label
Dim pMixedFontLabel As IMixedFontGridLabel
Set pMixedFontLabel = New MixedFontGridLabel
'Set IMixedFontGridLabel properties
Dim pFont As IFontDisp
Set pFont = New StdFont
pFont.Name = "Arial"
pFont.size = 12
pMixedFontLabel.SecondaryFont = pFont
pMixedFontLabel.SecondaryColor = BuildRGB(0, 0, 0)
pMixedFontLabel.NumGroupedDigits = 6 '-1 if not being used
'Set IFormattedGridLabel properties
Dim pFormattedGridLabel As IFormattedGridLabel
Set pFormattedGridLabel = pMixedFontLabel
Dim pNumericFormat As INumericFormat
Set pNumericFormat = New NumericFormat
pNumericFormat.AlignmentOption = esriAlignRight
pNumericFormat.RoundingOption = esriRoundNumberOfDecimals
pNumericFormat.RoundingValue = 2
pNumericFormat.ShowPlusSign = True
pNumericFormat.UseSeparator = False
pNumericFormat.ZeroPad = True

```



A measured grid with mixed font labels

Index grid tab styles



The index grid tab style coclasses provide the means to label an index grid. These coclasses are described below.

IndexGridTabStyle : IUnknown	Provides access to members that control the way an index grid's labels are drawn.
ForegroundColor: IColor	The foreground color of the tab.
OutlineColor: IColor	The outline color of the tab.
Thickness: Double	The thickness of the tab in points.
PrepareDraw (in labelValue: String, in tabWidthPage: Double, in axis: esriGridAxisEnum)	Sets up the tab for drawing.

The *IIndexGridTabStyle* interface provides access to the color and thickness of the index grid's labels. The *PrepareDraw* method should be called before *IGridLabel::Draw* is called on index grid tab style labels.

You can create an index grid tab style label using a coclass that inherits from *IndexGridTabStyle*, as outlined in the following examples. The code illustrates how to populate the properties exposed by the *IIndexGridTabStyle* interface after you create the label:

[Visual Basic 6.0]

```
' Create colors
Dim pForegroundColor As IColor
Dim pOutlineColor As IColor
Set pForegroundColor = New RGBColor
Set pOutlineColor = New RGBColor
```

```

pForegroundColor.RGB = &HECDEFE ' -> Pink
pOutlineColor.RGB = &HBBBBBB ' -> Gray

'Set IIndexGridTabStyle properties
pIndexGridTabStyle.ForegroundColor = pForegroundColor
pIndexGridTabStyle.OutlineColor = pOutlineColor
pIndexGridTabStyle.Thickness = 20

```

Button tab style labels are rectangular buttons, each the width of the grid cell that it borders. The following code shows you how to create a button tab style grid label.

[Visual Basic 6.0]

```

'Create the label
Dim pIndexGridTabStyle As IIndexGridTabStyle
Set pIndexGridTabStyle = New ButtonTabStyle

```

Continuous tab style labels form a continuous band around the map grid. The example below shows how you can create a label of this kind:

[Visual Basic 6.0]

```

Dim pIndexGridTabStyle As IIndexGridTabStyle
Set pIndexGridTabStyle = New ContinuousTabStyle

```

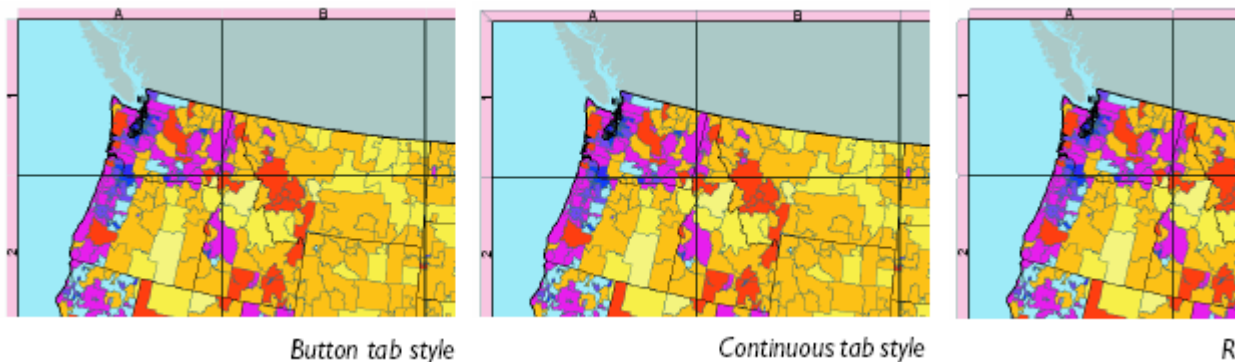
Rounded tab style labels are rounded rectangles; each one is the width of the grid cell it borders. Using the example below, you can create your rounded tab style grid label.

[Visual Basic 6.0]

```

Dim pIndexGridTabStyle As IIndexGridTabStyle
Set pIndexGridTabStyle = New RoundedTabStyle

```



A background tab style labels the index grid using square, round, or rounded-square boxes. These boxes are centered outside the grid cells they border.

IBackgroundTabStyle : IUnknown	Provides access to members that control background tab style grid labels.
BackgroundType: esriBackgroundTabType	The type of the background tab style.

IBackgroundTabStyle has a *BackgroundType* property you can use to determine the shape of the boxes that the *BackgroundTabStyle* label uses using the *esriBackgroundTabType* enumeration.

Enumeration esriBackgroundTabType	Types of background tabs for index grids.
0 - esriBackgroundTabRound	Round.
1 - esriBackgroundTabRectangle	Rectangle.
2 - esriBackgroundTabRoundedRectangle	Rounded rectangle.

The example below illustrates how you can create a background tab style label that uses round boxes to label a map grid.

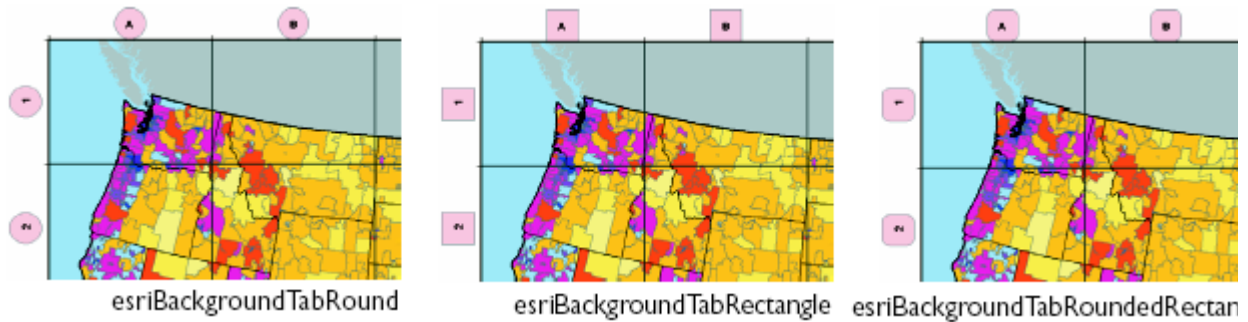
[Visual Basic 6.0]

```

Dim pIndexGridTabStyle As IIndexGridTabStyle
Set pIndexGridTabStyle = New BackgroundTabStyle
'Set IBackgroundTabStyle properties
Dim pBackgroundTabStyle As IBackgroundTabStyle
Set pBackgroundTabStyle = pIndexGridTabStyle

```

```
pBackgroundTabStyle.BackgroundType = esriBackgroundTabRound
```



Renderers

The ArcGIS renderer coclasses and interfaces are found mainly in the Carto library. Renderers are objects that store symbolization for ArcGIS data layers and draw these data based on the stored symbolization rules. Layer coclasses and interfaces are also mainly in the Carto library. The renderer objects are divided into three main groups: Feature renderers--objects that render feature data, raster renderers--objects that render raster data, and TIN renderers--objects that render 3D TIN data.

Renderers are assigned to layers using different interfaces depending on whether you are working with feature, raster, or TIN data. See the sections covering the different renderer groups below for details. After changing a layer's renderer properties, or after assigning a new renderer to a layer, be sure to update the display and table of contents to reflect these changes:

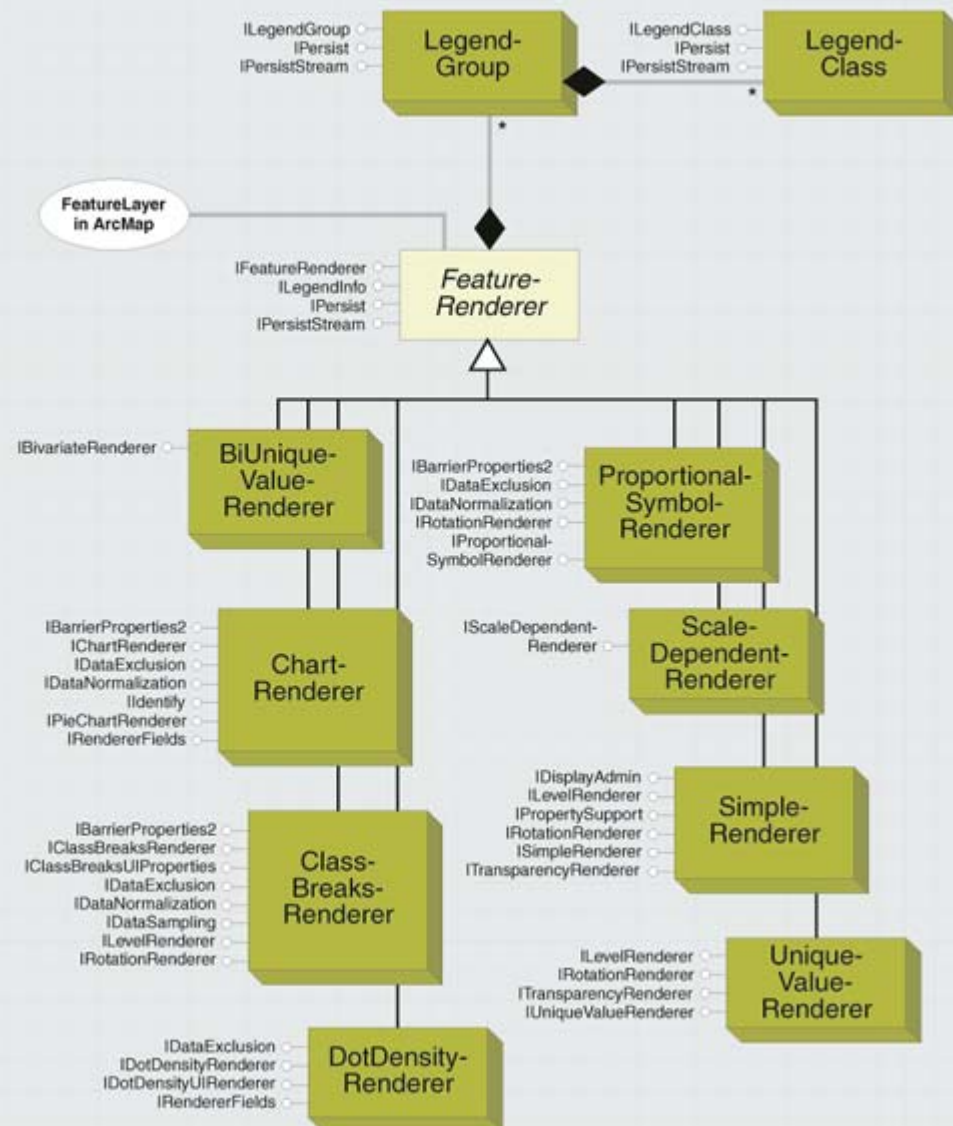
[Visual Basic 6.0]

```
Dim pDoc as IMxDocument
pDoc.ActiveView.PartialRefresh esriDPGeography, pLayer, Nothing
pDoc.UpdateContents
```


Most renderers store symbol objects and use these to draw layers. Symbol coclasses and interfaces are mainly in the [Display](#) library. To assign or retrieve a renderer's symbols, use the specific interface for your particular renderer. For example, if you are working with a *ClassBreaksRenderer*, then use *IClassBreaksRenderer::Symbol*, for a *RasterUniqueValueRenderer* use *IRasterUniqueValueRenderer::Symbol*, or for a *TinElevationRenderer* use *ITinColorRampRenderer::Symbol*. See the sections covering the different renderer groups below for details.


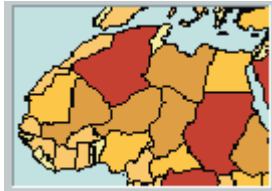
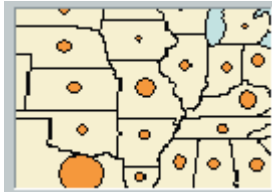
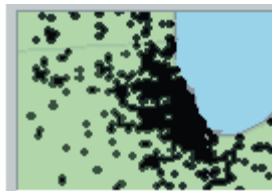

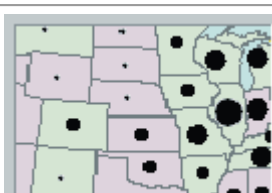
Feature Renderers

Feature renderer objects



Feature renderers are used to symbolize feature layers. A number of different feature renderer coclasses are available:

Feature renderer	Use	Map example
<i>SimpleRenderer</i>	Each feature in the layer is symbolized with the same symbol	
<i>UniqueValueRenderer</i>	Features are drawn with different symbols depending on the unique values of one or more attribute fields	

		
<i>ClassBreaksRenderer</i>	Features are drawn with graduated color (choropleth) or graduated size symbolization based on values in an attribute field. The data value range is divided into classes, each feature having membership in only one of these classes based on its attribute value. Features are drawn with the symbol corresponding to their class	
<i>ProportionalSymbolRenderer</i>	Features are drawn with proportionally sized symbols based on values in an attribute field	
<i>DotDensityRenderer</i>	Polygon features are drawn with randomly placed dots where the dot count for each feature represents the value of an attribute field	
<i>ChartRenderer</i>	Features are drawn with pie, bar, and stacked bar charts. Chart content is based on values stored in one or more attribute fields	
<i>BiUniqueValueRenderer</i>	Features are drawn with a symbolization that is a combination of a <i>UniqueValueRenderer</i> and a <i>ClassBreaksRenderer</i> . Values from separate, orthogonal attribute fields generate a single symbolization based on these two components	
<i>ScaleDependentRenderer</i>	This is a renderer that is composed of multiple renderers, each operates only within a particular scale range	

Accessing feature renderers

Each feature layer is assigned only one feature renderer. To get a feature renderer object from a layer, use *IGeoFeatureLayer::Renderer*. To set a renderer to a layer use the same property. After this assignment, be sure to also associate the correct renderer property page so that users can work with renderer properties on the Layer Properties dialog > Symbology tab. To associate a renderer property page, use *IGeoFeatureLayer::RendererPropertyPageClassID*. One way to do this is to use the *UID* object and give it the GUID of the property page. The *ProgID* can also be used, though it is not guaranteed to be unique:

[Visual Basic 6.0]

```
Dim pUID as New UID ' create a new UID objects
'progID is "esriCore.BarChartPropertyPage"
pUID.Value = "{98DD7040-FEB4-11D3-9F7C-00C04F6BC709}"
pGeoFeatureLayer.RendererPropertyPageClassID = pUID
```

FeatureRenderer abstract class

All feature renderer coclasses inherit from an abstract class *FeatureRenderer*. Developers can implement their own feature renderer by implementing all of the interfaces implemented by this class.

FeatureRenderer implements *IFeatureRenderer* which provides access to properties and methods common to all feature renderers. This base class also implements persistence interfaces that allow feature renderers to be saved and loaded from disk as part of .lyr and .mxd files. All feature renderers should also implement *ILegendInfo* which provides access to a renderer's legend information and symbols. All ESRI feature renderers have at least one *LegendGroup* which in turn has at least one *LegendClass*. Though feature renderers typically store all of the symbols they use to draw features in their *LegendClass* objects, it is best to access these symbols via the specific renderer interface for the renderer object you are using. For example, for a *ClassBreaksRenderer* use *IClassBreaksRenderer::Symbol*, or for a *ProportionalSymbolRenderer* use *IProportionalSymbolRenderer::MinSymbol* and *IProportionalSymbolRenderer::BackgroundSymbol*. Also note that though *ILegendInfo* is also implemented by most layer coclasses, layers typically defer all methods and properties to the renderer's implementation.

Creating a custom feature renderer and property page

To get complete control over the drawing of features in a layer you can write a custom renderer. See the [PointDispersalRenderer](#) example in Extending ArcObjects for a detailed walkthrough for creating a custom feature renderer. Here you are also taken through the steps to write a custom feature renderer property page which is the best way to fully integrate your renderer with the ArcGIS framework.

You can also associate your custom renderer with a particular feature class in the geodatabase by writing a feature class extension. For details, see the section in Extending ArcObjects on [Managing Custom Feature Renderers](#)

The ArcObjects developer samples also contain a number of custom feature renderer examples.

Raster Renderers

Raster renderers are used to symbolize raster layers. A number of different raster renderer coclasses are available:

Raster renderer	Use
<i>RasterClassifyColorRampRenderer</i>	Pixels are drawn with graduated color symbolization based on their value. The data value range is divided into classes, each pixel having membership in only one of these classes based on its value. Pixels are drawn with the color corresponding to their class
<i>RasterColormapRenderer</i>	The raster layer is drawn based on a color map
<i>RasterRGBRenderer</i>	This renderer is for drawing raster layers in true-color RGB. The renderer draws three bands of a raster dataset, one for each of the red, green, and blue channels
<i>RasterStretchColorRampRenderer</i>	This renderer is for rasters with continuous values. The raster layer is drawn using a color ramp stretched across these values
<i>RasterUniqueValueRenderer</i>	Pixels are drawn with a different color for each unique value in the raster layer

Accessing raster renderers

Each raster layer is assigned only one raster renderer. To get a raster renderer object from a layer use *IRasterLayer::Renderer*. To set a raster renderer to a layer use the same property. In the case where you have a *RasterCatalogLayer*, use *IRasterCatalogLayer::Renderer* instead.

RasterRenderer abstract class

All raster renderer coclasses inherit from an abstract class *RasterRenderer*. Developers can implement their own raster renderer by implementing all of the interfaces implemented by this class.

RasterRenderer implements *IRasterRenderer* which provides access to properties and methods common to all raster renderers. This base class also implements persistence interfaces that allow raster renderers to be saved and loaded from disk as part of .lyr and .mxd files. Raster renderers implement *ILegendInfo* which provides access to a renderer's legend information and symbols. ESRI raster renderers typically store the symbols they use to draw raster data in *LegendClass* objects which can be accessed via *ILegendInfo*. However, when writing client code that works with a raster renderer, it is best practice to access and change a renderer's symbols using the interface specific to the renderer you are using. For example, if working with a *RasterUniqueValueRenderer* use *IRasterUniqueValueRenderer::Symbol*, or if you are using a *RasterStretchColorRampRenderer*, use *IRasterStretchColorRampRenderer::ColorRamp*. Also note that though *ILegendInfo* is also implemented by most layer coclasses, layers typically defer all methods and properties to the renderer's implementation.

All raster renderers also expose display properties and methods through *IDisplayAdmin* and *IRasterDisplayProps*.

TIN Renderers

TIN renderers are used to symbolize TIN layers. A number of different TIN renderer coclasses are available. Each TIN layer can be symbolized by more than one of these objects at the same time. The ESRI TIN renderer objects are divided into renderers that symbolize nodes, edges, and faces.

TIN renderer	Use
<i>TinNodeRenderer</i>	Draws TIN nodes
<i>TinNodeValueRenderer</i>	Draws TIN nodes to show node values
<i>TinNodeRenderer</i>	Draws TIN nodes
<i>TinNodeElevationRenderer</i>	Draws TIN nodes to show elevation
<i>TinEdgeRenderer</i>	Draws TIN edges
<i>TinBreaklineRenderer</i>	Draws TIN edges to show breaklines
<i>TinFaceRenderer</i>	Draws TIN faces
<i>TinFaceValueRenderer</i>	Draws TINs at face value
<i>TinElevationRenderer</i>	Draws TIN faces to show elevation
<i>TinSlopeRenderer</i>	Draws TIN faces to show slope
<i>TinAspectRenderer</i>	Draws TIN faces to show aspect

Accessing TIN renderers

Two TIN renderers are added by default when you load a TIN layer, or a new layer is created, in ArcMap or ArcScene. In ArcMap the default renderers are a *TinEdgeRenderer* and a *TinElevationRenderer*, while in ArcScene (at ArcGIS 9.0 and later versions), the default renderers are a *TinEdgeRenderer* and a *TinFaceRenderer*. In versions prior to ArcGIS 9.0, ArcScene creates a *TinEdgeRenderer* and a *TinElevationRenderer* by default.

A TIN layer keeps a collection of TIN renderers, and you can get these use *ITinLayer*. The following code reports the name of each renderer used by a TIN Layer:

[Visual Basic 6.0]

```
Public Sub RendererReport()
    Dim pMxDoc As IMxDocument
    Dim pMap As IMap
    Dim pTinLayer As ITinLayer
    Dim pTinRend As ITinRenderer
    Dim i As Integer
    Set pMxDoc = Application.Document
    Set pMap = pMxDoc.FocusMap
    Set pTinLayer = pMap.Layer(0)
    For i = 0 To pTinLayer.RendererCount - 1
        Set pTinRend = pTinLayer.GetRenderer(i)
        MsgBox pTinRend.Name
    Next
End Sub
```

You can add multiple renderers, even of the same general type -- node, edge, and face. -- to a TIN layer's renderer collection. However, when two ore more renderers of the same type are added, there can be a conflict, and it is the order in which you add your renderers that resolves this conflict. Also, in the ArcGIS user interface -- TIN layer properties dialog > symbology tab -- users can toggle on/off the various TIN renderers assigned to a layer.

TinRenderer abstract class

All TIN renderer coclasses inherit from an abstract class *TinRenderer*. Developers can implement their own TIN renderer by implementing all of the interfaces implemented by this class.

TinRenderer implements *ITinRenderer* which provides access to properties and methods common to all TIN renderers. This base class also implements persistence interfaces that allow TIN renderers to be saved and loaded from disk as part of .lyr and .mxd files. TIN renderers also implement *ILegendInfo* which provides access to a renderer's legend information and symbols. ESRI TIN renderers typically store the symbols they use to draw TIN data in *LegendClass* objects which can be accessed via *ILegendInfo*. However, when writing client code that works with a TIN renderer, it is best practice to access and change a renderer's symbols using the interface specific to the renderer you are using. For example, if working with a *TinNodeValueRenderer* use *ITinUniqueValueRenderer*, or if you are working with a *TinSlopeRenderer*, then use *ITinColorRampRenderer::Symbol*. Also note that though *ILegendInfo* is also implemented by most

layer coclasses, layers typically defer all methods and properties to the renderer's implementation.

Labeling

One of the key factors in creating a usable map is labeling (or annotating) features on the map. Labeling is the placing of text near a feature to purvey information about that feature. Normally the label is based on attribute values of the feature itself, but it doesn't have to be.

The ArcGIS labeling environment offers a wide variety of methods for labeling features and for resolving conflicts when labels overlap each other. The labeling environment includes the ability to specify which features are to be labeled (all features, features identified by an SQL query, and so on); the expression that is used to label them (expressions can be simple or complex based on VBScript and JScript); placement options and weights for those placements; and priority specifications of one layer versus another. Depending on the requirements of the user, it is also possible to label one layer with multiple expressions.

The objects in this model provide the ability to access all of the parameters associated with the labeling of features. Advanced developers can also create their own expression-parsing engines to be used in the labeling process.

The *AnnotateLayerPropertiesCollection* holds a collection of the different labeling sets (*LabelEngineLayerProperties* objects) assigned to a particular feature layer. The collection can be created, or it can be retrieved from the *IGeoLayer::AnnotationProperties* property on a feature layer. It is possible to label a layer with more than one expression. The purpose of the *AnnotateLayerPropertiesCollection* object is to keep track of the set of expressions that have been assigned.

The *IAnnotateLayerPropertiesCollection* interface allows for the manipulation of the *IAnnotateLayerProperties* (*LabelEngineLayerProperties* coclass) objects held within the collection. Through the interface, the developer can add, remove, sort, and query the objects in the collection.

QueryItem provides access to the items in the collection as well as the placed and unplaced elements that go with each *LabelEngineLayerProperties* object.

A *LabelEngineLayerProperties* object maintains the set of properties associated with the labeling of a feature layer. Multiple *LabelEngineLayerProperties* can be created for a single feature layer; they are stored within an *AnnotateLayerPropertiesCollection*. The object keeps track of which features to label, how to label them, what symbology to use, how to transform the labels based on the current scale, and what to do with unplaced labels.

The *IAnnotateLayerProperties* interface is implemented only by the *LabelEngineLayerProperties* object and provides the answer to the question of which features to label and at what scales. Through this interface, the developer can specify the priority of the labels, a where clause to be applied to the feature layer, and a specification of what to do with unplaced elements.

The *FeatureLinked*, *LabelWhichFeatures*, and *GraphicsContainer* properties apply only when the set of labels is being converted to annotation. The developer can use the *GraphicsContainer* property to specify where the converted labels will go.

The *FeatureLayer* property is used internally during the labeling process. If you find it necessary to set this property, be sure to set it back to Null after labeling has completed.

The *IAnnotateLayerTransformationProperties* interface is implemented only by the *LabelEngineLayerProperties* object; it holds the settings that determine what size to draw the labels at different scales. Use this interface when you want to specify the reference scale and other transformation properties to use with a *LabelEngineLayerProperties* object.

The *ScaleRatio* is a ratio between the *IAnnotateLayerProperties::Extent* property and the *IAnnotateLayerTransformationProperties* property.

The *ILabelEngineLayerProperties* interface is implemented by the *LabelEngineLayerProperties* object and provides access to the expression, symbol, and overposting properties of the label engine object. Use this interface when you want to access the *AnnotationExpressionEngine* and *BasicOverposterLayerProperties* objects associated with the label engine object.

By default, the *ExpressionParser* property will return the *AnnotationVBScriptEngine* object. In general, the developer would not use this property unless they wanted to use Java scripting for labeling. In this case, an *AnnotationJScriptEngine* object would be created, and the *ExpressionParser* property would be set to this. The expression to use is always set through the *Expression* property.

The *IsExpressionSimple* property identifies whether a complex expression is being used in the *Expression* property. Complex expressions involve a parser object (*ExpressionParser* property) to parse the string.

The *SymbolID* property is used during the conversion of labels to annotation when a symbol collection in the *AnnotationFeatureClassExtension* is referenced.

More precise control of the labeling properties can be obtained by accessing the *BasicOverposterLayerProperties* of the *LabelEngineLayerProperties* object. You can either create the *BasicOverposterLayerProperties* object, or you can retrieve it from the *ILabelEngineLayerProperties::BasicOverposterLayerProperties* property.

The *IBasicOverposterLayerProperties* interface is implemented only by the *BasicOverposterLayerProperties*

object—it provides access to the overposting resolution methods employed by the label engine object. Each set of labeling properties defines how conflict resolution (overposting) issues will be resolved for those labels. The *IBasicOverposterLayerProperties* interface provides access to most of these properties.

FeatureType specifies whether labeling is being performed on point, line, or polygon features. Be sure to check this property before accessing any feature-type-specified property, such as *LineLabelPosition* or *PointPlacementAngles*.

FeatureWeight specifies whether labels can be placed on top of the features in a layer, while *LabelWeight* specifies whether the labels can conflict with other labels. The *NumLabelsOption* indicates how many labels to place per feature.

The *IBasicOverposterLayerProperties2* interface is implemented by the *BasicOverposterLayerProperties* object and was added to allow you to set the maximum distance a label could be placed from its target (*MaxDistanceFromTarget* property). This property is only used when the label engine is used to place chart symbols. The *IBasicOverposterLayerProperties3* interface was added to provide access to point rotation labeling capabilities. The *IBasicOverposterLayerProperties4* interface was added to provide access to polygon labeling properties.

The *IOverposterLayerProperties* interface is implemented by the *BasicOverposterLayerProperties* object and provides access to whether labels or symbols are placed. The label engine only places symbols when it is used to place chart symbols at which point these properties are controlled by the chart renderer.

The *IsBarrier* property indicates whether the features in the layer should serve as barriers for label placement (do not put labels on top of the features).

Annotation

Annotation in ArcGIS is enabled through the use of a feature class extension. The *AnnotationFeatureClassExtension* is used to configure the drawing properties and symbology for annotation features. Annotation feature classes are created using methods on the *IAnnotationLayerFactory* interface on the *FDOGraphicsLayerFactory* object.

The annotation feature class extension

Annotation features persist and draw text or graphic elements stored in the geodatabase. An annotation feature class (*AnnoClass*) can be feature-linked or standalone. Feature-linking allows the text of the annotation to be derived from the value of a related feature. The lifetime of the annotation is also controlled by the lifetime of the related feature. The *IAnnotationClassExtension* interface is used to access the properties of the annotation feature class extension. Commonly accessed properties are *AnnotationLayerProperties* and *SymbolCollection*.

Annotation features can persist (store) either an entire symbol inline or reference a symbol in a symbol collection. These two persistence mechanisms balance performance with flexibility.

Storing the symbol inline allows the modification of the symbol on a feature-instance basis. Unfortunately, this method increases the size of a row dramatically and may cause performance degradation when drawing large numbers of features in a multiuser environment.

A more efficient but less flexible alternative is to use symbols in the *SymbolCollection* of the *AnnotationFeatureClassExtension*. There symbols are stored as a property of the *AnnotationFeatureClassExtension*. The annotation feature stores an ID that references a symbol in the extension's *SymbolCollection*. *TextElements* are linked to symbols in the symbol collection by using the *ISymbolCollectionElement* interface on the *TextElement*. A small number of commonly changed attributes can be overridden with no minimal performance penalties using the *ISymbolCollectionElement* interface. Once an annotation feature has an element with a collection symbol, it is important that the symbol is not removed or modified in the *SymbolCollection*.

The *IAnnoClassAdmin3* interface is used to update the properties of the class. In a versioned geodatabase, these properties apply to all versions and are not versioned. After creating an annotation feature class, modifying these properties may cause problems with the drawing and selection of annotation features. Adding new symbols to the *SymbolCollection* or changing the *AutoCreate*, *UpdateOnShapeChange*, *OverposterProperties*, and *RequireSymbolID* properties are the only recommended modifications. Deleting or modifying symbols in the *SymbolCollection* requires updating all annotation features whose elements reference the group symbol. When adding new symbols to the *SymbolCollection* use the *AddSymbol* method of *ISymbolCollection2*, which will automatically assign a unique ID to the symbol. As with any schema related change, it is recommended that an exclusive schema lock be obtained before calling the *UpdateProperties* method.

The *AllowSymbolOverrides* property indicates if an annotation may override a symbol property even though it references the symbol collection. If set to false, only the *AnchorPoint*, *Background*, and *TextPath* of the text symbol can be modified. Modifications of these properties should be performed using *ISymbolCollectionElement*. This property is best employed in conjunction with *RequireSymbolID* and is used to limit the properties of an annotation feature that can be edited for either database efficiency of institutional reasons.

The *OverposterProperties* property indicates which label engine is to be used by the annotation feature class to generate/update feature-linked annotation.

The *RequireSymbolID* property indicates if annotation features are required to reference a symbol in the symbol collection. Referencing a symbol in the symbol collection is the most efficient way to store annotation features. If a symbol does not reference a symbol in the symbol collection, the text symbol is inefficiently stored inline with the feature in the feature's blob field. Setting this property ensures that features reference a symbol in the symbol collection and are therefore efficiently stored. To avoid separating annotation features from their referenced symbol, modify the properties of each annotation element using *ISymbolCollectionElement*. These interfaces allow access to the properties of an annotation feature class that can be overridden. Note that setting *AllowSymbolOverrides* to False greatly reduces the number of properties that can be overridden. See above for more details.

The annotation feature

The *AnnotationFeature* persists and draws *GraphicElements* that are stored in the geodatabase. For labeling, a *TextElement* is used. Annotation features can be linked to features in a related *FeatureClass*.

The *IAnnotationFeature* interface is used for relating *AnnotationFeatures* to other features or updating the graphic of the annotation. The *Annotation* property accepts any *GraphicElement*. If a *TextElement* is used, a group symbol can be assigned by using the *IGroupSymbol* interface. A *TextElement* that does not reference a symbol in the symbol collection will have a group symbol ID of -1.

To relate an *AnnotationFeature* to another feature (a *RelationshipClass* must already exist), assign the OID of the related feature to the *LinkedFeatureID* property. If the *AnnotationFeature* is not linked, the *LinkedFeatureID* property is -1. After updating either of these properties, the *IFeature::Store* methods must be called.

The *IAnnotationFeature2* interface was added to provide access to the *AnnotationClassID* and *Status* of the annotation feature. You may wish to update the status of an annotation feature from "Unplaced" to "Placed" after altering it.

Dimensions

Dimensions are a special kind of map annotation that show specific lengths or distances on a map. A dimension may indicate the length of a side of a building or land parcel or the distance between two features such as a fire hydrant and the corner of a building. Dimensions can be as simple as a piece of text with a leader line or more elaborate. In the geodatabase, dimensions are stored in dimension feature classes.

Like other feature classes in the geodatabase, all features in a dimension feature class have a geographic location and attributes and can either be inside or outside of a feature dataset. Like annotation features, each dimension feature knows what its symbology is and how it should be

Dimensions in ArcGIS are enabled through the use of a feature class extension. The *DimensionClassExtension* is used to configure the drawing properties and symbology for annotation features. Annotation feature classes are created using methods on the *IAnnotationLayerFactory* interface on the *FDOGraphicsLayerFactory* object.

The dimension class extension

The *IDimensionClassExtension* interface provides access to the *DimensionStyles* collection and the reference scale drawing properties. The *ReferenceScale* property defines the scale at which symbols are drawn (at their defined size).

The *ReferenceScaleUnits* property is only used when the *DimensionFeatureClass*' spatial reference is Unknown. Changing the *ReferenceScale* after the feature class contains features is not recommended, as those features' geometries are controlled by the *ReferenceScale* property.

After making changes to any of the *IDimensionClassExtension* properties, it is necessary to call the *UpdateProperties* method. Changes can also be discarded by calling the *ResetProperties* method if *UpdateProperties* has not been called. As with any schema-related modification, an exclusive schema lock should be obtained on the feature class before calling *UpdateProperties*.

Dimension styles

The *DimensionStyles* coclass is used to retrieve, create, and delete the *DimensionStyles* coclass.

The *IDimensionStyles* interface provides methods and properties for managing *DimensionStyle* objects.

In order to add a new *DimensionStyle* object, create a new *DimensionStyle* coclass, modify it, and call the *Add* method. When a style is added, a *StyleID* is automatically assigned to that *Style*.

The *DefaultStyleID* property specifies which style should be used by default in ArcMap. *DimensionStyle* objects can be retrieved by ID or name using the *GetStyle* and *FindStyle* methods.

Existing *DimensionStyle* objects can be renamed using the *Rename* method. Styles can only be deleted and not modified.

If a *DimensionStyle* is deleted, it is important to reassign a new *DimensionStyle* to existing *DimensionFeatures* that reference the deleted style.

The *DimensionStyle* coclass supports three interfaces for managing the symbology, behavior, and text of a

dimension.

The *IDimensionStyle* interface provides properties for identifying *DimensionStyle* coclass.

The *ID* property is read-only; it is assigned to a *DimensionStyle* when it is added to a *DimensionStyles* collection.

The *Name* property provides a label for the style and is set before adding a *DimensionStyle* to the *DimensionStyles* collection. The *Name* property must be unique within a *DimensionStyles* collection.

The *IDimensionStyleDisplays* interface is used to control the display properties of the various parts of a dimension.

The *esriDimensionDisplay* enumeration defines four values for use with several properties.

The *MarkerFit* property controls a dimension's behavior for fitting the text and label.

The *esriDimensionMarkerFit* enumeration defines three values.

Setting *MarkerFit* to *esriDimensionMarkerFitTolerance* moves markers outside of extension lines if the *MarkerFitTolerance* is exceeded. Setting *MarkerFit* to *esriDimensionMarkerFitText* moves markers to the outside if colliding with text. This option does not apply to custom text positions.

When the markers are moved because of a fit, a line will be drawn between the markers based on the *DrawLineOnFit* property. The *BaselineHeight* property specifies the height above the selected dimension at which new dimensions will be created when using the Baseline Dimension tool in ArcMap.

The *IDimensionStyleText* interface contains properties that control how the text of a dimension is displayed. The *Align* property forces the text to align to the angle of the *DimensionLine*. If the *Align* property is *False*, the *TextSymbol's* angle is used.

The *ConvertUnits* property specifies whether or not the value of the text will be converted from the *FeatureClass's* native units to the units of the *DisplayUnits* property.

The text can be formatted using the *DisplayPrecision* property and the *TextDisplay* property. The *esriDimensionTextDisplay* enumeration defines four values for formatting the text string.

The text string can also be determined from an expression specified in the *Expression* property. The expression can be a simple concatenation of column values and strings or a function written in scripting language.

The name of the parser for the expression is specified in the *ExpressionParserName* property. The currently available parsers are "VBScript" and "JScript". The *TextFit* property determines where the text will be placed if it does not fit between the markers after they have been moved (due to marker fit settings).

The *esriDimensionTextFit* enumeration defines three values for this behavior.

When the markers are moved because of a fit, a line will be drawn between the markers based on the *DrawLineOnFit* property.

The dimension feature

The *DimensionFeature* coclass draws dimensions using a *DimensionStyle* coclass. The feature's *DimensionShape* determines the placement and length of the dimension.

The *IDimensionFeature* provides properties for setting the style and placement of a *DimensionFeature*.

The *StyleID* property should be a valid ID from the class' *DimensionStyles* collection. If the current ID is invalid, the *DimensionFeature* will draw its boundary in red.

The *DimensionShape* property defines the placement of the elements of a dimension. The location and size of the *DimensionFeature* are determined entirely by the *DimensionShape*; it is not necessary to use the *IFeature::Shape* property.

The *DimensionType* property defines the type of the dimension as linear or aligned and affects how the Edit tool behaves with the *DimensionFeature* during shape modification.

The *DimensionLineDisplay*, *ExtensionLineDisplay*, and *MarkerDisplay* properties are values that override the values of the current *DimensionStyle* coclass.

A custom value for the *DimensionFeature's* text can be set using the *CustomLength* property and by setting the *UseCustomLength* property to *True*.

The *DimensionShape* coclass stores points for a dimension's measurements. The *DimensionFeature* and *DimensionGraphic* use *DimensionShapes* to draw and store dimensions.

The *IDimensionShape* interface supports properties for the definition of a dimension's location and measurement.

The *BeginDimensionPoint* and *EndDimensionPoint* properties define the dimension's measurement point.

The *DimensionLinePoint* property determines the height of the dimension line above the baseline.

To create a two-point dimension, the *DimensionLinePoint* must be the same value as the *BeginDimensionPoint*.

The *ExtensionLineAngle* property defines the angle between the dimension line and the extension line in degrees. The default angle is 90 degrees; oblique dimensions have angles less than or greater than 90

degrees.

The *DimensionShape* supports a custom text location using the *TextPoint* property.

For the default location of the dimension text, the *TextPoint's IGeometry::IsEmpty* property should be True.

The dimension graphic

The *DimensionGraphic* is used for dynamically rendering dimensions using a *DimensionStyle* and *DimensionShape*.

The *IDimensionGraphic* interface provides methods and properties for drawing dimensions.

The *Style* property sets the *DimensionStyle* for the *DimensionGraphic*. The *DimensionShape* defines the location of the dimension's measurements and text.

The *Length* property returns the current calculated length for the dimension. A custom length value can be specified using the *CustomLength* and *UseCustomLength* properties.

If the current *DimensionShape* contains a nonempty *TextPoint*, the default location for the text is available through the *GetDefaultTextPoint* method.

Layers

Layers display geographic information on a map. A layer doesn't store the actual geographic data; rather it references the data contained in coverages, shapefiles, geodatabases, images, grids, and so on, then defines how to display this geographic data. Some layers do not refer to geographic data. For example, a *GroupLayer* refers to other layers, and a *CompositeGraphicsLayer* stores graphics.

Each different type of layer object represents different types of data. Examples of layer objects include *FeatureLayer*, *GraphicsLayer*, *RasterLayer*, *TinLayer*, *CoverageAnnotationLayer*, and *GroupLayer*. The third page of the [Carto Object Model Diagram](#) is a good place to start for an understanding of the different layer classes and the properties and methods that they support.

The following is a brief description of each layer coclass and class in the Carto library:

Layer	Description
<i>CadAnnotationLayer</i>	Displays CAD annotation
<i>CadFeatureLayer</i>	Displays CAD data as feature data
<i>CadLayer</i>	Displays CAD data as a CAD drawing
<i>CompositeGraphicsLayer</i>	A collection of graphics layers that behaves like a single layer
<i>CoverageAnnotationLayer</i>	Displays ArcInfo Workstation coverage annotation, PCArcInfo Coverage annotation, or SDE3 annotation
<i>DimensionLayer</i>	Displays geodatabase dimension features
<i>DummyGraduatedMarkerLayer</i>	Displays graduated marker legend items in the style gallery
<i>DummyLayer</i>	Displays legend items in the style gallery
<i>FDOGraphicsLayer</i>	Displays geodatabase annotation features
<i>FDOGraphicsSublayer</i>	Displays annotation features in a subset of a geodatabase annotation feature class
<i>FeatureLayer</i>	Displays vector features, for example stored in geodatabase feature classes, shapefiles, and coverages
<i>GdbRasterCatalogLayer</i>	Displays raster catalog data stored in a geodatabase
<i>GraphicsSubLayer</i>	Graphic sublayer handed back by a <i>CompositeGraphicsLayer</i>
<i>GroupLayer</i>	A collection of layers that behaves like a single layer
<i>IMSMapLayer</i>	Displays IMS data
<i>MapServerLayer</i>	Displays ArcGIS Map Server data
<i>RasterCatalogLayer</i>	Displays raster catalog data
<i>RasterLayer</i>	Displays raster data
<i>TinLayer</i>	Displays TIN data
<i>Topology</i>	Displays a geodatabase topology as a layer

Accessing layers

The *Map* object manages a collection of layers. You can use the *Layer* or the *Layers* property on the *IMap* interface to get a reference to a layer. To determine the type of layer to which you have a reference, query for specific interfaces. For example, if the layer object supports the *ITinLayer* interface, then you know it is a *TinLayer* object.

Layers can be persisted separately from maps in layer files (.lyr). The following code accesses a *FeatureLayer* object from a layer file. (Note that working with layer files is not supported with ArcGIS Engine, ArcGIS Desktop is required):

[Visual Basic 6.0]

```
Dim pGxCat As IGxCatalog, pGxLayer As IGxLayer
Dim pGxObj As IGxObject, pEnumGxObj As IEnumGxObject
Dim num As Long
Dim pValue As Variant
Set pGxCat = New GxCatalog
Set pValue = pGxCat.GetObjectFromFullName("d:\mydata\states.lyr", num)
If TypeOf pValue Is IEnumGxObject Then
    Set pEnumGxObj = pValue
    Set pGxObj = pEnumGxObj.Next
Else
    Set pGxObj = pValue
End If
Set pGxLayer = pGxObj
Dim pGFLayer As IGeoFeatureLayer
Set pGFLayer = pGxLayer.Layer
```

Accessing a layer's underlying data

Depending on the type of layer you are working with, you use a different interface to access the layer's underlying data source. In some cases this interface is different for each type of layer, for example for a *RasterLayer* use *IRasterLayer::Raster* to access the underlying data for the layer. For layers that refer to feature data, you can use *IGeoFeatureLayer::FeatureClass* to access the underlying feature class for the layer. A *GroupLayer* has no direct data source, and instead it refers to other layers. Access these layers via *ICompositeLayer*.

To learn more, see the ArcObjects component help for your particular layer.

Layer abstract class

All layers inherit from an abstract class called *Layer*. The *Layer* class implements several interfaces: *ILayer*, *IGeoDataset*, *ILayerGeneralProperties*, *IPublishLayer*, *IPersist*, and *IPersistStream*.

The *ILayer* interface has a method to draw the layer and properties to define the extent of the layer, the minimum and maximum display scale, the spatial reference, the name, the supported draw phases, and the map tip text. There are also properties that indicate whether the layer is visible, valid, or cached, and whether or not the layer shows map tips. The *Cached* property indicates whether the layer requires its own display cache or not. If *Cached* is set to True, the *Map* will give a separate display cache to the layer so it can be refreshed independently of all other layers. A tracking layer is a good example of a custom layer that would set the *Cached* property to True. Note that the *SpatialReference* property is used only for the map display; it does not change the spatial reference of the underlying data. It carries the *Map* object's knowledge of the current on-the-fly projection back to the feature layer.

IGeoDataset specifies the extent and spatial reference of the underlying data. The *SpatialReference* property on *IGeoDataset* is read-only. The property is used to set the spatial reference of the *Map*; the *Map*'s spatial reference is automatically set to the spatial reference of the first layer loaded.

ILayerGeneralProperties provides access to a text description for the layer and also properties for the previous maximum and minimum visibility scales for the layer. *IPublishLayer* provides access to properties and methods for publishing the layer using the ArcGIS Publisher extension. The persistence interfaces allow a layer to be saved and loaded from disk as part of a .lyr or .mxd file.

To learn more about layer interfaces and coclasses see the ArcObjects component help.

Creating a custom layer

You can write a custom layer to support an unsupported data type in ArcGIS, or to change how a supported data type draws and behaves in ArcMap and the other ArcGIS applications. To learn more about writing a custom layer, see the section [Creating Custom Layers](#) in Extending ArcObjects.

MapServer

MapServer is a coarse-grained ArcObjects component allowing users to display and query ArcGIS map documents bound to a *MapServer* object. *MapServer* can be used in either desktop, intranet (LAN/WAN), or internet development environments.

In a desktop environment, *MapServer* objects can be created in-process within your application. You can use *MapServer*, as well as other coarse-grained objects as a "short-cut" for ArcObjects development. Operations that took many lines of code may now just take a few.

More typically, however, *MapServer* objects are run within ArcGIS Server. When running in ArcGIS Server, *MapServer* objects can be accessed via the [Server API](#) over TCP/IP (intranet), or can be accessed over HTTP using SOAP/XML (internet).

Web applications and Web services

MapServer objects can be consumed by server-based Web applications. ArcGIS Server provides an Application Developer Framework (ADF) for both .NET and Java. Web applications run on Web servers that are on the same LAN as a GIS server and access server objects via the server API. This allows the developer to create a wide range of GIS web applications using ArcObjects. The ADF includes a set of Web templates, Web controls and ArcObjects proxies.

ArcGIS Server also includes a SOAP/XML toolkit that allows a *MapServer* object to be exposed as a Web service (HTTP connection). The Web service consumer can get the methods and types exposed by the *MapServer* Web service through its Web Service Description Language (WSDL). WSDL describes the definitions of SOAP requests and responses. They are part of the GIS server and are installed as part of the ArcGIS Server install under <install directory>\XMLSchema.

The following *MapServer* interfaces do not support SOAP/XML: *IMapServerData*, *IMapServerLayout*, and *IMapServerObjects*.

ArcGIS Desktop and ArcGIS Engine

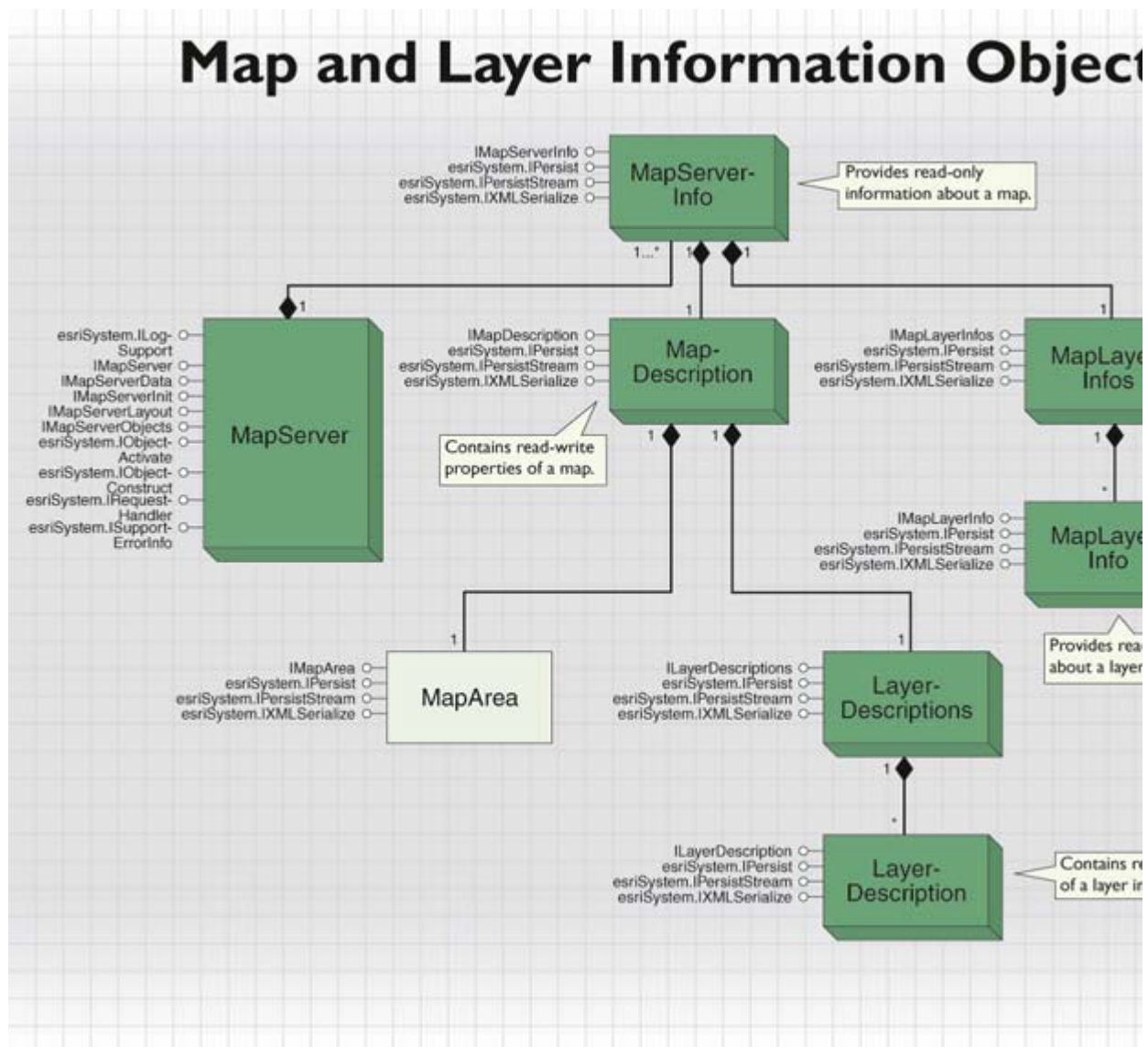
ArcGIS Desktop and ArcGIS Engine developers can consume ArcGIS Server Web services using the ArcGIS Server Client API (*AGSClient*). *AGSClient* includes objects and methods for connecting to GIS servers either directly (TCP/IP) or through Web service catalogs (HTTP). *AGSClient* differs from the Server API in that it restricts clients to calling only the coarse-grained methods on the server object and does not provide access to the fine-grained ArcObjects associated with the server object. The *MapServer* interfaces *IMapServerData*, *IMapServerLayout* and *IMapServerObjects* are not supported by *AGSClient*.

ArcGIS Desktop and ArcGIS Engine developers can also consume server objects via the Server API. If you want to work with a server object in a stateful manner, or you want to work with the fine-grained ArcObjects, you must use the Server API. You can use the *GIServerConnection* to connect to the GIS server, or you can obtain a reference to the *ServerObjectManager* through *GISClient* (TCP/IP connection only).

The *MapServer* coclass

The *MapServer* coclass contains several interfaces with basic functions for displaying (*IMapServer* and *IMapServerLayout*) and querying (*IMapServer* and *IMapServerData*) an ArcGIS map document (.mxd or .pmf). Also, there are a number of associated *MapServer* objects that represent input and output parameters for methods on *MapServer* interfaces. For example, the *IMapServer* method *ExportMapImage* requires two inputs: a description of the map to be exported and a description of the output parameters. These inputs are captured in the *MapDescription* and *ImageDescription* objects.

Information about the map and its layers



To access information about a map and its layers use the *IMapServer* method *GetServerInfo*. *GetServerInfo* returns a *MapServerInfo* object. Use the *IMapServerInfo* interface to access read-only information about a map (data frame). *IMapServerInfo* provides access to members describing the default state of a *MapServer* object, such as the name of the map, the background color or the spatial extent of the map. This information cannot be changed without changing the state of the underlying fine-grained *ArcObjects*.

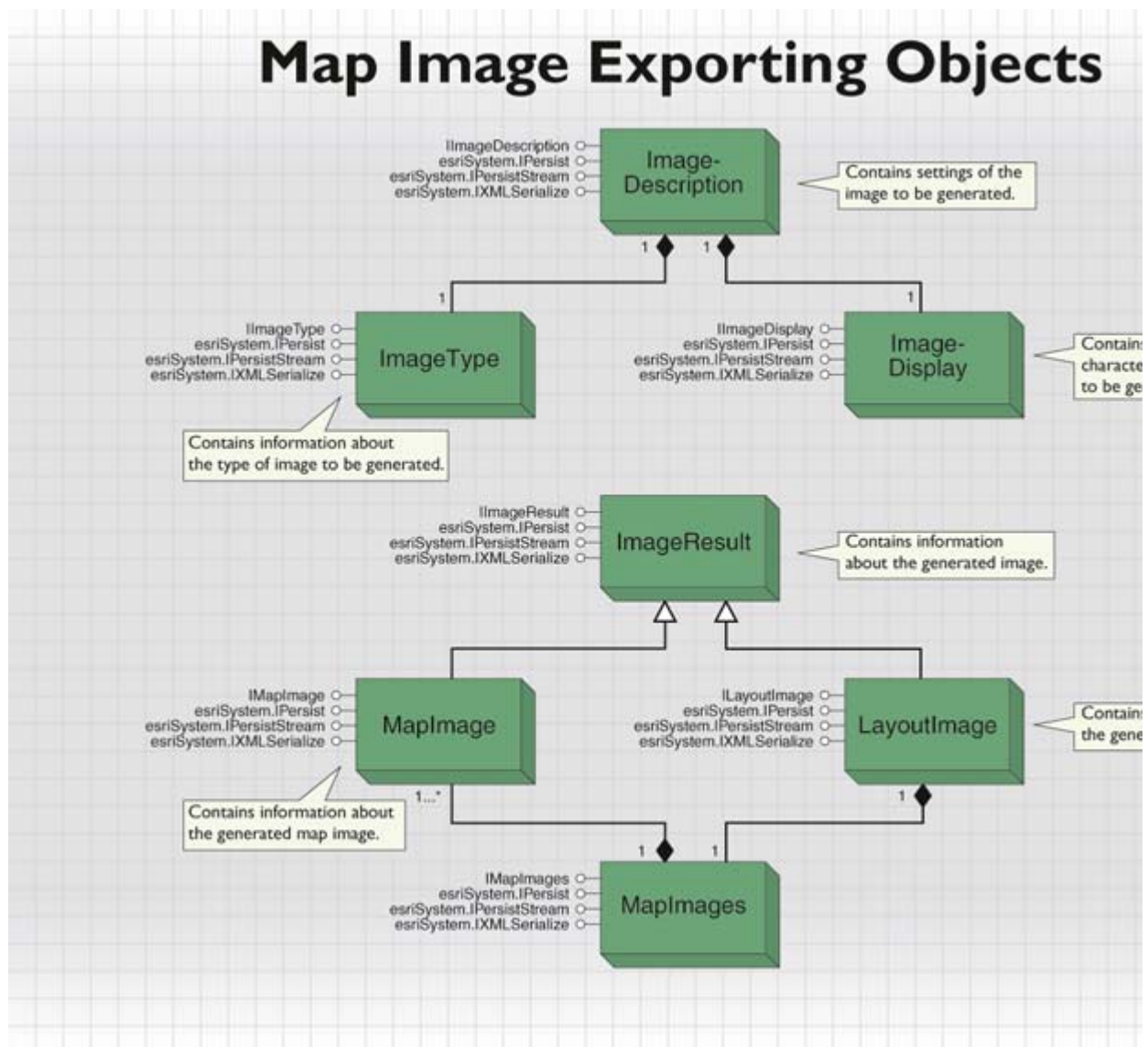
A *MapServerInfo* object contains a *MapDescription*. Use *IMapDescription* to access map settings that can be changed on the server object without changing the state of the underlying fine-grained *ArcObjects* that the map document is based on.

Please note the difference between *IMapServerInfo* and *IMapDescription*. *IMapServerInfo* provides read-only access to its members. *IMapDescription* provides read-write access to its members. Use *IMapDescription* to access and change map settings without changing the state of the fine-grained *ArcObjects*.

Layer settings can be retrieved from the *IMapLayerInfo* and *ILayerDescription* interfaces. Use the *IMapLayerInfo* interface to access read-only information about an individual layer in the map. Use the *ILayerDescription* interface to access read and write properties of a layer.

Please note the difference of use between *IMapLayerInfo* and *ILayerDescription*: Use *ILayerDescription* to access layer settings that can be changed on the server object without changing the state of the underlying fine-grained *ArcObjects* that the layer is based on. *MapLayerInfo* is used to retrieve information about a layer that can only be changed by directly accessing the map document or the fine-grained *ArcObjects* it is based on.

Exporting a map image

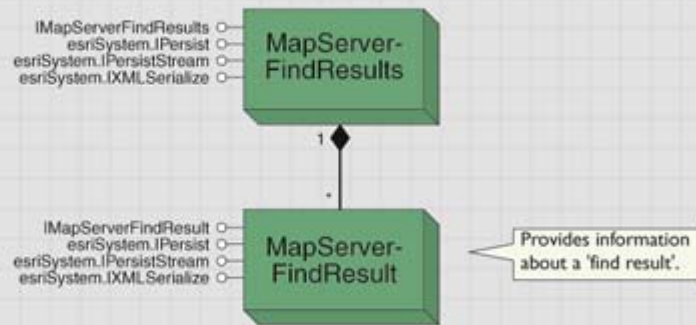


To export a map image use the method *ExportMapImage* on the interface *IMapServer*. To specify the size and type of the output image, use the *IImageDescription*, *IImageDisplay* and *IImageType* interfaces. *MapServer* supports all ArcGIS supported output types. *ExportMapImage* returns a *MapImage* object. The interfaces *IMapImage*, along with *ILayoutImage* inherit from *IImageResult*.

Querying the map

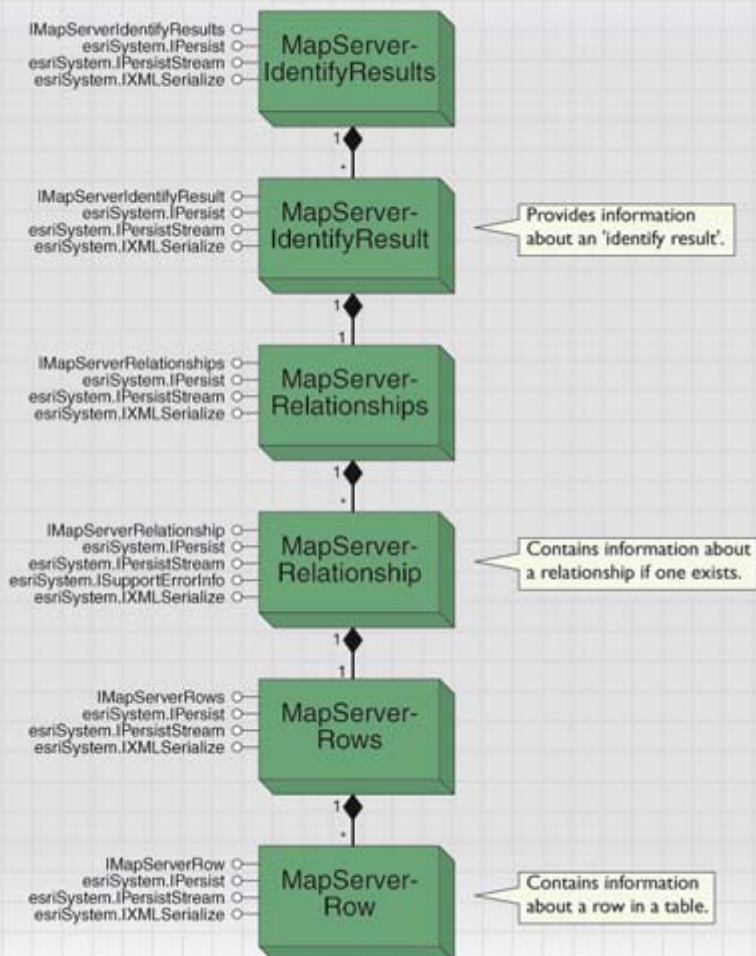
To perform query operations on the map, *IMapServer* offers a number of methods. These include: *Find*, *Identify*, *QueryHyperlinks*, *QueryFeatureCount*, *QueryFeatureData*, and *QueryFeatureIDs*. In order to control the amount of information *MapServer* needs to process for a query, a maximum number of records can be set. This value is contained in the *MaxRecordCount* property on *IMapServerInit*. It can be set there, or if the *MapServer* object is being used in a server context, the *MaxRecordCount* can be changed by modifying the *MaxRecordCount* XML tag in the *MapServer*'s configuration file. The default value for this property is 500. This setting does not affect *QueryFeatureCount* or *QueryFeatureIDs*.

MapServer Find Result Objects



The *Find* method returns a *MapServerFindResults* object. This is a collection of *MapServerFindResult* objects. Use *IMapServerFindResult* to access properties of found features.

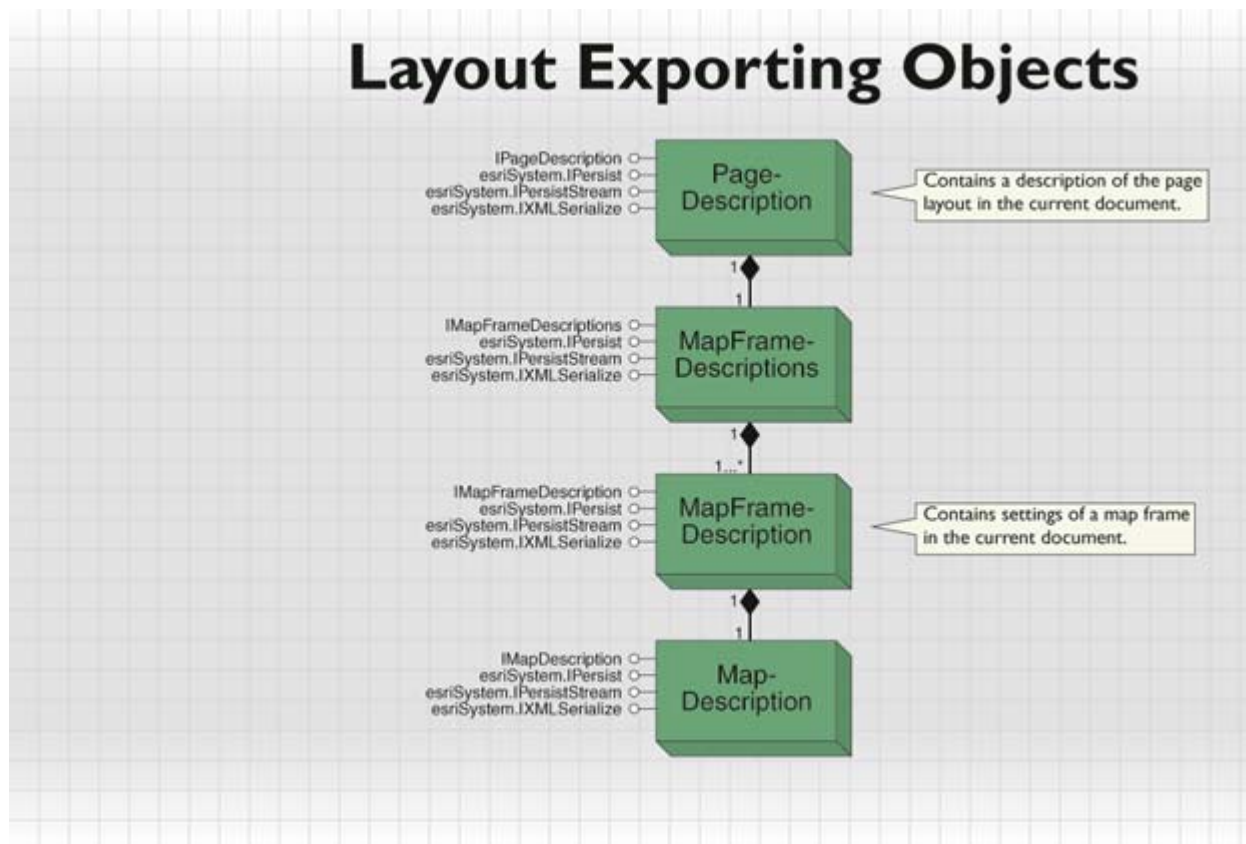
MapServer Identify Result Object



Identify returns a *MapServerIdentifyResults* object. This is a collection of *MapServerIdentifyResult* objects. Use *IMapServerIdentifyResult* to access properties of identified features. This includes rows associated to the feature through a table relationship.

Map layouts and map surrounds

Use the *IMapServerLayout* interface to export an existing map layout or to export a legend, North arrow or a scale bar of an existing map. Also use *IMapServerLayout* to convert screen coordinates to page coordinates on the layout and vice versa.



One of the requirements for the *ExportLayout* method on *IMapServerLayout* is a *PageDescription* object. The *PageDescription* contains a *MapFrameDescriptions* object, which is the collection of *MapFrameDescriptions* (data frames) present in the layout.

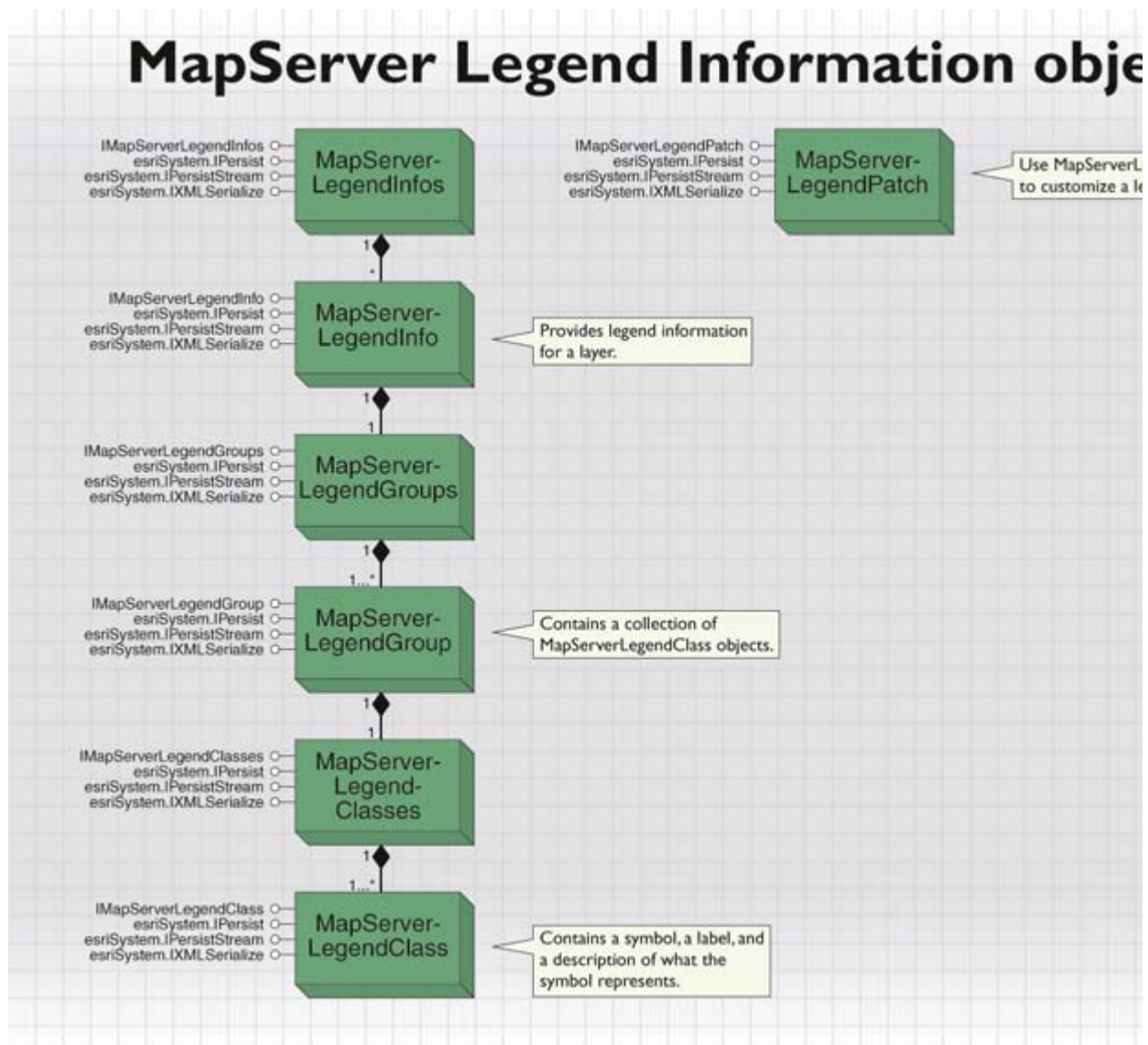
Some layout elements can change dynamically when accessed by *IMapServerLayout*. Data frames in the layout will reflect any changes in *MapDescription*. Scale bars, scale text, North arrows and legends linked to data frames will adjust to these changes.

Do not use *IMapServerLayout* to create new layouts or to create layouts "on-the-fly". You can not change the actual page size of the map layout, reposition map elements on the layout nor can you add or remove layout elements such as titles, text, graphics, scale bars, North arrows and legends. You must access finer-grained *ArcObjects* via *IMapServerObjects* to make these kinds of changes.

Please note that *IMapServerLayout* can neither be accessed through the ArcGIS Server Client API (*AGSClient*) nor through SOAP/XML.

Information about individual legend elements

In addition to exporting a single image of the legend using *IMapServerLayout*, you can retrieve individual legend elements including the symbol images, labels, descriptions and headings. A common use would be to populate a table of contents. To retrieve this legend information, use *GetLegendInfo* on *IMapServer*. This method returns a *MapServerLegendInfos* object, which is a collection of *MapServerLegendInfo* objects. Using these objects you can retrieve only the parts of the legend you are interested in.



Accessing fine-grained ArcObjects

Though the methods and properties available through *MapServer* and its associated objects offer important mapping functionality, they cannot possibly encapsulate all that ArcObjects offers. In many cases you may want to use other, finer-grained, ArcObjects in conjunction with *MapServer*. You can do this using the *IMapServerObjects* interface. Through this interface you can access *ILayer*, *IMap* and *IPageLayout*. For example, you can make changes to the map, such as adding a new layer, using *IMap*.

It is very important to distinguish between temporary and permanent changes to the *MapServer* object. A temporary change would include changes to the *MapDescription* or *LayerDescription* using *IMapDescription* and *ILayerDescription*. For example, you might change the geographic extent of a map (*MapArea*) or change the visibility of a layer (*Visible*). These changes can be temporary and valid for the duration of the call. Once the call has ended the *MapServer* object returns to its default state.

Permanent changes to the *MapServer* object can be done in three ways. The first is by changing the map document itself and then restarting the *MapServer* object. The second is by changing *MapServer* properties using such interfaces as *IMapDescription* and *ILayerDescription* and then calling the *IMapServerObjects* method *ApplyMapDescription*. This will update the state of the *MapServer* object. The third way to change the *MapServer* object is to access the underlying fine-grained ArcObjects directly using the methods on *IMapServerObjects*, make a change such as adding a new layer or changing a layer's rendering, and then calling *RefreshServerObjects*. This refreshes the *MapServer* object with the current state held by the fine-grained ArcObjects.

Please note that *IMapServerObjects* can neither be accessed through the ArcGIS Server Client API (AGSClient) nor through SOAP/XML. Calls into finer-grained ArcObjects must be done in a desktop environment or over a TCP/IP connection using the Server API.

ArcIMS layers, symbols, and renderers

The Internet is a vast resource for geographic data. Organizations can publish their data using ArcIMS and serve it over the Internet. The Geography Network, which also uses ArcIMS, provides easy access to data on the Internet. You can view this data as layers in ArcMap. ArcIMS provides two types of map services: an ArcIMS Feature Service and an ArcIMS Image Service.

An ArcIMS feature service is similar to a feature dataset that contains many feature classes. Each ArcIMS feature class represents a unique entity; the actual features are streamed to the client. When you add a feature service to the map, a group layer consisting of one or more feature layers is also added to the map. You can work with feature layers based on ArcIMS feature services in ArcMap the same way you work with feature layers based on local feature classes.

An ArcIMS image service is a raster representation of a complete map. When you add an image service to ArcMap, you'll see a new layer on your map. This layer is an Internet Map Server map layer (*IMSMapLayer*). You can turn off specific sublayers in the IMS map layer so you see only those that are of interest to you.

An *IMSMapLayer* is a composite layer consisting of IMS sublayers. You can use the *ICompositeLayer::Layer* property to get a reference to an IMS sublayer; the sublayer is of type *IIMSSubLayer*. From that, you can get a reference to the *ACLayer* (Arc Connection layer) on which the sublayer is based. An *ACLayer* does not implement *ILayer*; rather, it is an XML representation of the layer from the Internet service. *ACLayers* use the symbology defined on the Internet service for display in ArcMap.

The *IIMSMapLayer* interface indicates that the layer is an IMS map layer. The *Connection* property and *Connect* method manage the connection to the Internet service. The *MoveSubLayerTo* method allows you to rearrange the order of the sublayers in the *IMSMapLayer*. The *IMSMap* property returns a reference to the *ACMap* (Arc Connection map), which is an XML representation of the map that was served over the Internet.

A series of ArcIMS symbol and renderer object exist in the Carto library to aide in the management of the display of *IMSMapLayers*. The objects maintain the relationship between the XML and display representations of the data. The *IMSMapLayer* manages this relationship and there is generally no need to access or modify these objects.

GPS Support

The ArcMap GPS Support coclasses and interfaces are located mainly in the Carto library. A few classes and interfaces, including *GpsExtension* can be found in the ArcMapUI library. The main coclass is the *RealTimeFeedManager*. Through it, you can access and control either a realtime feed from a GPS device or a simulated feed from a feature class. Use *GpsFeed* together with *GpsConnection* to control the connection to a GPS device. If you want to replay data logged earlier or other appropriate data, use *RealTimeFeedSimulator*. Both *GpsFeed* and *RealTimeFeedSimulator* support the *IRealTimeFeed* interface. All information on the current position can be accessed through this interface.

The *RealTimeFeedManager* provides access to *IGpsDisplayProperties* where you can customize how positions are displayed. The *Show* properties such as *ShowCurrentPosition* turn on or off various attributes of the current position or the marker trails including the altitude, bearing, or speed. You can set the minimum and maximum altitude and speed values plus set the minimum and maximum sizes of the symbols used for the altitude values. The symbology of the speed is set through *SpeedColorRamp*. *RealTimeFeedManager* has methods to refresh or clear the GPS display.

The GPS toolbar supports a Destination and you can control the Destination properties with *IRealTimeDestination*. The *ShowBearingToDestination* and *BearingToDestinationSymbol* add a symbol to the current position that points to the active destination. The *DestinationLocation* defines the position of the destination while the *DestinationSymbol* defines which symbol to use. You can also control the *DestinationLabel* and *DestinationTextSymbol*.

You can display previous positions with markers or a linear trail with *IPositionTrails*. The *LinearTrailDistance* and *LinearTrailSymbol* are the properties of a linear trail while *MarkerTrailDistance*, *MarkerTrailCount*, *MarkerTrailSymbol* and *MarkerTrailColorRamp* control various features of a marker trail. Turn trails on or off with the *ShowMarkerTrails* and *ShowLinearTrails* boolean properties. You can store positions in a feature class through the *IRealTimeLog* interface. The *StartLogging* and *StopLogging* methods turn logging on and off, respectively, while the *StampGpsPosition* method will write the current position to the log file. The *LogFile* properties defines the feature class to which positions will be logged. If you are streaming position to the log file, use the *LogRate*, *MinimumLogDeflectionAngle*, or the *MinimumLogDistance* to control which locations are written to the log file.

If you wish to snap the current position to an existing layer, use *IRealTimeFeedSnap*. You can set the snapping distance, the layers to snap to, and whether to snap to nodes, lines, or vertices.

The code below illustrates how to get a reference to the GPS toolbar and write the current position to a point geometry.

[Visual Basic 6.0]

```
'Get a pointer to the GPS extension
Dim pUID As esriSystem.IUID
Set pUID = New esriSystem.IUID
pUID.Value = "{C994BFE6-47F1-4BAC-8F35-0743C7576673}"
```

```
Set m_GPSExt = m_App.FindExtensionByCLSID(pUID)

'Put the GPS location into a point in memory
Dim pPt As esriGeometry.IPoint
Set pPt = New esriGeometry.Point
pPt.PutCoords m_GPSExt.RealTimeFeedManager.RealTimeFeed.CurrentPosition.longitude, _
    m_GPSExt.RealTimeFeedManager.RealTimeFeed.CurrentPosition.latitude
```