CHAPTER 8

GEOMETRIC ROUTINES AVENUE WRAPS

This chapter contains Avenue Wraps that enable the programmer to (a) create and/or retrieve geometric features such points, multipoints, lines and polygons, and (b) to intersect, merge or union two feature shapes. These Avenue Wraps include the following:

◗	**avPolylineMake**	To create a polyline object from a point list as per Figure 8-2	8-19
◗	**avPolylineMake2**	To create a polyline object from a point list as per Figure 8-1	8-19
◗	**avRectMake4Pt**	To create a polygon given the coordinates of the four corners comprising the rectangle	8-20
◗	**avRectMakeXY**	To create a polygon given the coordinates of two opposite corners of a rectangle	8-21
◗	**avReturnArea**	To get the area of a geometry	8-24
◗	**avReturnCenter**	To get the centroid of a geometry	8-24
◗	**avReturnDifference**	To subtract one shape from another shape to form a new shape	8-29
◗	**avReturnIntersection**	To intersect two shapes to form a new shape	8-29
◗	**avReturnLength**	To get the length of a geometry	8-25
◗	**avReturnMerged**	To merge two shapes to form a new shape	8-30
◗	**avReturnUnion**	To union two shapes to form a new shape	8-31
◗	**avSplit**	To split a shape using a second shape as a splitter	8-31
◗	**avUnion**	To union two or more geometry objects to form a new geometry object	8-32
◗	**GetShape**	To create a list containing the coordinates of the points that comprise a feature	8-12
◗	**Sample Code**	Code examples on how to perform various shape editing operations	8-33

The source listing of each of the above Avenue Wraps may be found in Appendix D of this publication.

GENERAL GEOMETRY

8.1 General Geometric Avenue Wraps

8.1.1 Subroutine avaClassMake

This subroutine enables the programmer to create a point, polyline or polygon type geometry object given a collection of points (shapeList) and the desired object type (aClass). In using this subroutine, note the following:

1. The given argument aClass specifies the desired object type, and its numeric value indicates the following object type to be created:

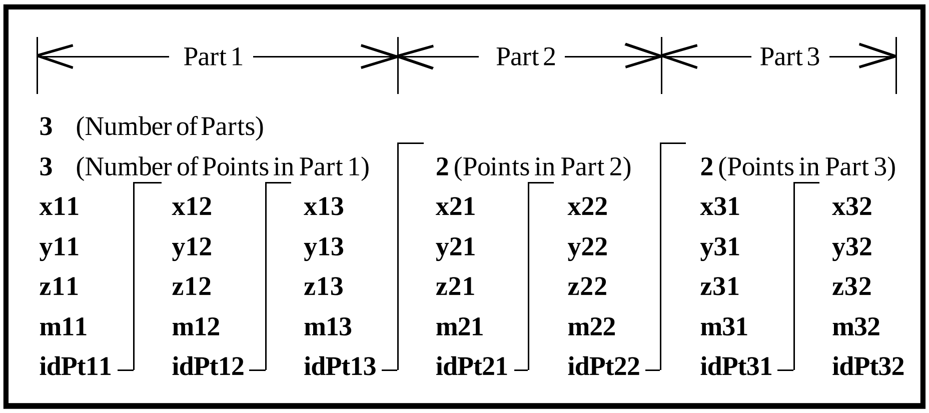
   11 for PolyLineM
   12 for PolyLineZ
   13 for PolygonM
   14 for PolygonZ
   15 for PointM
   16 for PointZ
   17 for MultiPointM
   18 for MultiPointZ
   31 for PolyLineM and PolyLineZ
   32 for PolygonM and PolygonZ
   33 for PointM and PointZ
   34 for MultiPointM and MultiPointZ
   41 for PolyLine
   42 for Polygon
   43 for Point
   44 for MultiPoint

2. The given argument shapeList is a collection comprised of the following items: nParts / nPoints / xPt / yPt / zPt / mPt / idPt

   where: nParts / nPoints / idPt are declared as long integer numbers denoting the number of parts, number of points in a part and identification number of a point,

   and: xPt / yPt / zPt / mPt are declared as double precision floating numbers denoting the x, y and z coordinates, and the measure of a point.

3. As an example of the composition of the shapeList collection, consider a multi-point, polyline or polygon comprised of three parts, the first having three points, the second two points and the third two points. The



Part 1			Part 2		Part 3	
3 (Number of Parts)						
3 (Number of Points in Part 1)			**2** (Points in Part 2)		**2** (Points in Part 3)	
x11	**x12**	**x13**	**x21**	**x22**	**x31**	**x32**
y11	**y12**	**y13**	**y21**	**y22**	**y31**	**y32**
z11	**z12**	**z13**	**z21**	**z22**	**z31**	**z32**
m11	**m12**	**m13**	**m21**	**m22**	**m31**	**m32**
idPt11	**idPt12**	**idPt13**	**idPt21**	**idPt22**	**idPt31**	**idPt32**

Figure 8-1 Composition of Sample shapeList Collection - avaClassMake

contents of shapeList would then be as shown in Figure 8-1, with the first suffix indicating the part, and the second indicating the point number.

The corresponding Avenue request is:

There is no corresponding Avenue request.

avaClassMake

The call to this Avenue Wrap is:

Call **avaClassMake**(aClass, shapeList, theFeat)

GIVEN:	aClass	= the type of special feature. See Note 1 above.
	shapeList	= the list of points comprising the feature
RETURN:	theFeat	= the special feature

The given and returned variables should be declared where first called as:

Dim aClass As Integer
Dim shapeList As New Collection
Dim theFeat As IPoint, IMultiPoint, IPolyline, or IPolygon

8.1.2 Subroutine avAsList

This subroutine enables the programmer to create a collection of points that comprise a point, polyline, polygon or multi-point using the **IFeature** interface. The composition of the resultant collection (shapeList) is shown in Figure 8-2, which represents a collection of collections of points, each of last said collections representing a part of the specified feature (theFeature).

The corresponding Avenue request is:

shapeList = theFeature.AsList

avAsList

The call to this Avenue Wrap is:

Call **avAsList**(theFeature, shapeList)

GIVEN:	theFeature	= feature to be processed
RETURN:	shapeList	= the shape's list of list of points

The given and returned variables should be declared where first called as:

Dim theFeature As IFeature
Dim shapeList As New Collection

GENERAL GEOMETRY

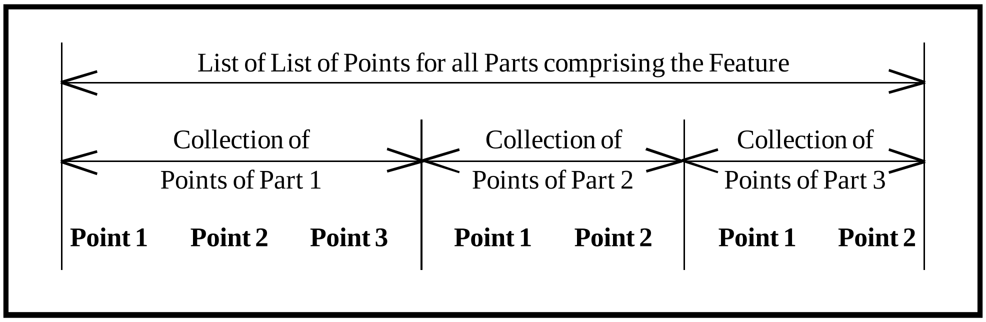

List of List of Points for all Parts comprising the Feature
Collection of Points of Part 1
Collection of Points of Part 2
Collection of Points of Part 3
Point 1
Point 2
Point 3
Point 1
Point 2
Point 1
Point 2

Figure 8-2 Composition of Sample shapeList Collection - avAsList

8.1.3 Subroutine avAsList2

This subroutine enables the programmer to create a collection of points that comprise a polyline or polygon feature. The composition of the resultant collection (shapeList) is shown in Figure 8-3(a).

The corresponding Avenue request is:
There is no corresponding Avenue request.

The call to this Avenue Wrap is:
Call **avAsList2**(theFeature, shapeList)

avAsList2

GIVEN: theFeature = feature to be processed

RETURN: shapeList = the shape's list of points and/or parts

The given and returned variables should be declared where first called as:
Dim theFeature As IFeature
Dim shapeList As New Collection

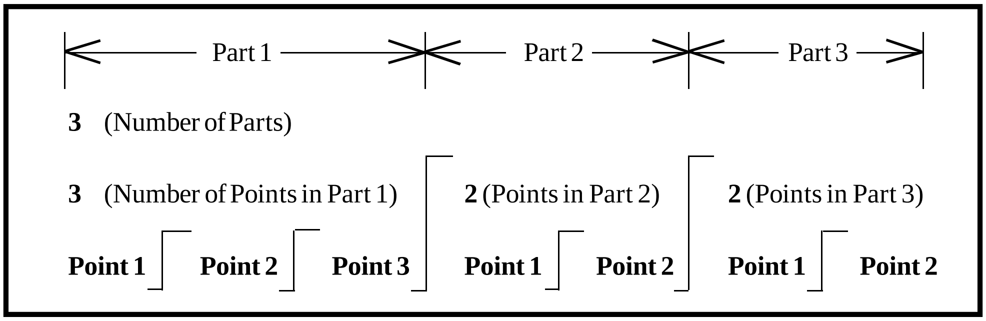


Figure 8-3(a) Composition of Sample shapeList Collection - avaClassMake

GENERAL GEOMETRY

8.1.4 Subroutine avAsList3

This subroutine enables the programmer to create a collection of points that comprise a polyline or polygon geometry object. The composition of the resultant collection (shapeList) is shown in Figure 8-3(a).

The corresponding Avenue request is:
There is no corresponding Avenue request.

The call to this Avenue Wrap is:

avAsList3

Call **avAsList3**(theGeometry, shapeList)

GIVEN: theGeometry = feature to be processed

RETURN: shapeList = the shape's list of points and/or parts

The given and returned variables should be declared where first called as:
Dim theGeometry As IGeometry
Dim shapeList As New Collection

8.1.5 Subroutine avPlAsList

This subroutine enables the programmer to create a collection of points from a geometry object. The composition of the resultant collection (shapeList) is shown in Figure 8-2. This routine can be used for point, polyline, polygon and multi-point features. This subroutine is identical to avAsList except that is operates on an IGeometry object rather than an IFeature object.

The corresponding Avenue request is:
shapeList = pFeatureGeom.AsList

The call to this Avenue Wrap is:

avPlAsList

Call **avPlAsList**(pFeatureGeom, shapeList)

GIVEN: pFeatureGeom = geometry object to be processed

RETURN: shapeList = list of list of points comprising the geometry

The given and returned variables should be declared where first called as:
Dim pFeatureGeom As IGeometry
Dim shapeList As New Collection

GENERAL GEOMETRY

8.1.6 Subroutine avPlAsList2

This subroutine enables the programmer to create a collection of points from a geometry object. The composition of the resultant collection (shapeList) is shown in Figure 8-1. This routine can be used for point, polyline, polygon and multi-point features.

The corresponding Avenue request is:

There is no corresponding Avenue request.

The call to this Avenue Wrap is:

avPlAsList2

Call **avPlAsList2**(theLine, shapeList)

GIVEN: theLine = geometry object to be processed

RETURN: shapeList = list of points comprising the geometry

The given and returned variables should be declared where first called as:

Dim theLine As IGeometry
Dim shapeList As New Collection

8.1.7 Subroutine avPlFindVertex

This subroutine enables the programmer to find the vertex within a multi-point, polyline or polygon feature that matches a given location or is the closest to a given location.

The corresponding Avenue request is:

There is no corresponding Avenue request.

The call to this Avenue Wrap is:

avPLFindVertex

Call **avPlFindVertex**(ipmode, elmntList, X, Y, thePart, thePt)

GIVEN: ipmode = the mode of operation
0 : find the first vertex matching a location
1 : find the vertex closest to a location

elmntList = list of points comprising the feature, see Figure 8.1

X, Y = coordinates of the location nearest which a vertex is to be compared with

RETURN: thePart = the part of the polyline. Part numbers begin at zero (0), not one (1)

GENERAL GEOMETRY

thePt	=	the sequential point number (starting at 1) in the part (thePart) nearest to the given location (X,Y).

The given and returned variables should be declared where first called as:

```
Dim ipmode As Integer
Dim elmntList As New Collection
Dim X, Y As Double
Dim thePart, thePt As Long
```

8.1.8 Subroutine avPlGet3Pt

This subroutine enables the programmer to get the coordinates of three points from a specific part in a specified collection of a feature point. The composition of the given point collection (shapeList) is as shown in Figure 8.1. The Avenue Wrap avPlAsList2 may be used to extract this collection if it is not known by any other means.

The corresponding Avenue request is:
There is no corresponding Avenue request.

avPlGet3Pt

The call to this Avenue Wrap is:
Call **avPlGet3Pt**(shapeList, thePart, X1, Y1, XM, YM, X2, Y2)

GIVEN:	shapeList	=	list of points comprising the feature
	thePart	=	the part of the polyline. Part numbers begin at zero (0), not one (1)
RETURN:	X1, Y1	=	start point coordinates of part
	XM, YM	=	mid point coordinates of part
	X2, Y2	=	end point coordinates of part

The given and returned variables should be declared where first called as:

```
Dim shapeList As New Collection
Dim thePart As Long
Dim X1, Y1, XM, YM, X2, Y2 As Double
```

GENERAL GEOMETRY

8.1.9 Subroutine avPlModify

This subroutine enables the programmer to modify a specific part in a specified collection of a feature. The composition of the given point collection (shapeList) is as shown in Figure 8.1. The Avenue Wrap avPlAsList2 may be used to extract this collection if it is not known by any other means. In using this subroutine, note the following:

1. The new collection to be created (newList) does not replace the given collection ShapeList. ShapeList remains unchanged. To replace ShapeList with newList use the CopyList Avenue Wrap.
2. This subroutine has no effect on the graphic representation of the feature.

The corresponding Avenue request is:

There is no corresponding Avenue request.

The call to this Avenue Wrap is:

```
Call avPlModify(ipmode, shapeList, thePart, iPt, X, Y, Z, newList)
```

avPlModify

GIVEN: ipmode = mode of operation. Numeric value to denote:
0 = change coordinates of given point iPt
1 = insert new point after given point iPt
2 = delete given point

shapeList = list of points comprising the feature

thePart = the part of the polyline. Part numbers begin at zero (0), not one (1)

iPt = point number (starting at 1) in the part to be processed. If it is 0, then the last point in the part will be processed.

X, Y, Z = coordinates of the new point

RETURN: newList = new list of points comprising the feature

The given and returned variables should be declared where first called as:

```
Dim ipmode As Integer
Dim shapeListAs New Collection
Dim thePart, iPt As Long
Dim X, Y, Z As Double
Dim newList As New Collection
```

GENERAL GEOMETRY

8.1.10 Subroutine GetShape

This subroutine enables the programmer to obtain information describing a feature, which consists of (a) the feature type, (b) a list containing the coordinates of the points that comprise a feature and (c) the length of the feature.

The corresponding Avenue request is:
There is no corresponding Avenue request.

GetShape

The call to this Avenue Wrap is:

```
Call GetShape(elmntTheme, elmntRecrd, _
                    shapeType, shapeList, shapeDist)
```

GIVEN:	elmntTheme	= theme of the feature
	elmntRecrd	= shape's list of points and/or parts
RETURN:	shapeType	= shape type enumerator
	shapeList	= shape's list of points and/or parts
	shapeDist	= ArcView length of the shape in map units

The given and returned variables should be declared where first called as:

```
Dim elmntTheme As Variant
Dim elmntRecrd As Long
Dim shapeType As esriGeometryType
Dim shapeList As New Collection
Dim shapeDist As Double
```

8.2 Geometric Feature Creation Avenue Wraps

GEOMETRIC FEATURE CREATION

The routines in this section create the geometric attributes that comprise the indicated geometric feature only. They do not create the graphic representation of the feature.

8.2.1 Function avAsPolygon

This function enables the programmer to change an IUnknown polygon interface into an IGeometry polygon interface.

The corresponding Avenue request is:

There is no corresponding Avenue request.

The call to this Avenue Wrap is:

Set thePolygon = **avAsPolygon**(pInput)

avAsPolygon

GIVEN:	pInput	= the IUnknown polygon interface to be converted
RETURN:	thePolygon	= the IGeometry polygon interface

The given and returned variables should be declared where first called as:

```
Dim pInput As IUnknown
Dim thePolygon As IGeometry
```

8.2.2 Function avCircleMakeXY

This function enables the programmer to create a circle given the coordinates of its center and its radius.

The corresponding Avenue request is:

theCircle = Circle.Make (aPoint, aRadius)

accepts as input an object (aPoint) rather than the X and Y coordinates of the center point

The call to this Avenue Wrap is:

Set theCircle = **avCircleMakeXY**(xPt, yPt, rad)

avCircleMakeXY

GIVEN:	xPt, yPt	= X and Y coordinates of the circle's center
	rad	= radius of the circle
RETURN:	theCircle	= the curve feature

GEOMETRIC FEATURE CREATION

The given and returned variables should be declared where first called as:
Dim xPt, yPt, rad As Double
Dim theCircle As ICurve

8.2.3 Function avMultipointMake

This function enables the programmer to create a multipoint object from a list of points.

The corresponding Avenue request is:
theMultiPoint = MultiPoint.Make (aPntList)

avMultipointMake

The call to this Avenue Wrap is:
Set theMultiPoint = **avMultipointMake**(aPntList)

GIVEN:	aPntList	= list of point (IPoint) objects
RETURN:	theMultiPoint	= the multipoint feature

The given and returned variables should be declared where first called as:
Dim aPntList As New Collection
Dim theMultiPoint As IMultipoint

8.2.4 Function avPointIDMake

This function enables the programmer to create a point given its X and Y coordinates and assign a user-specified ID value to the point.

The corresponding Avenue request is:
There is no corresponding Avenue request.

avPointIDMake

The call to this Avenue Wrap is:
Set thePoint = **avPointIDMake**(xPt, yPt, anID)

GIVEN:	xPt	= X coordinate of point
	yPt	= Y coordinate of point
	anID	= ID value to be assigned to the point
RETURN:	thePoint	= the point feature

The given and returned variables should be declared where first called as:
Dim xPt As Double, yPt As Double, anID As Long
Dim thePoint As IPoint

GEOMETRIC FEATURE CREATION

8.2.5 Function avPointMake

This function enables the programmer to create a point given its X and Y coordinates.

The corresponding Avenue request is:

```
thePoint = Point.Make (xPt, yPt)
```

The call to this Avenue Wrap is:

Set thePoint = **avPointMake**(xPt, yPt)

avPointMake

GIVEN:	xPt	= X coordinate of point
	yPt	= Y coordinate of point
RETURN:	thePoint	= the point feature

The given and returned variables should be declared where first called as:

```
Dim xPt, yPt As Double
Dim thePoint As IPoint
```

8.2.6 Function avPointMMake

This function enables the programmer to create a point given its X and Y coordinates and assign a user-specified M value to the point.

The corresponding Avenue request is:

```
thePoint = PointM.Make (xPt, yPt, anM)
```

The call to this Avenue Wrap is:

Set thePoint = **avPointMMake**(xPt, yPt, anID)

avPointMMake

GIVEN:	xPt	= X coordinate of point
	yPt	= Y coordinate of point
	anM	= M value to be assigned to the point
RETURN:	thePoint	= the point feature

The given and returned variables should be declared where first called as:

```
Dim xPt As Double, yPt As Double, anM As Double
Dim thePoint As IPoint
```

GEOMETRIC FEATURE CREATION

8.2.7 Function avPointSetID

This function enables the programmer to assign a user-specified ID value to a point object. Note that the given point object is modified by this procedure.

The corresponding Avenue request is:
There is no corresponding Avenue request.

avPointSetID

The call to this Avenue Wrap is:
Set thePoint = **avPointSetID**(thePoint, anID)

GIVEN:	thePoint	= the point feature to be modified
	anID	= ID value to be assigned to the point
RETURN:	thePoint	= the modified point feature

The given and returned variables should be declared where first called as:

```
Dim thePoint As IPoint
Dim anID As Long
```

8.2.8 Function avPointSetM

This function enables the programmer to assign a user-specified M value to a point object. Note that the given point object is modified by this procedure.

The corresponding Avenue request is:
thePoint.SetM (anM)

avPointSetM

The call to this Avenue Wrap is:
Set thePoint = **avPointSetM**(thePoint, anM)

GIVEN:	thePoint	= the point feature to be modified
	anM	= M value to be assigned to the point
RETURN:	thePoint	= the modified point feature

The given and returned variables should be declared where first called as:

```
Dim thePoint As IPoint
Dim anM As Double
```

GEOMETRIC
FEATURE
CREATION

8.2.9 Function avPointSetZ

This function enables the programmer to assign a user-specified Z value to a point object. Note that the given point object is modified by this procedure.

The corresponding Avenue request is:

```
     thePoint.SetZ (anZ)
```

The call to this Avenue Wrap is:

     Set thePoint = **avPointSetZ**(thePoint, anZ)

avPointSetZ

GIVEN:	thePoint	= the point feature to be modified
	anZ	= Z value to be assigned to the point
RETURN:	thePoint	= the modified point feature

The given and returned variables should be declared where first called as:

```
Dim thePoint As IPoint
Dim anZ As Double
```

8.2.10 Function avPointZMake

This function enables the programmer to create a 3D point given its X, Y and Z coordinates.

The corresponding Avenue request is:

```
     thePoint = PointZ.Make (xPt, yPt, zPt)
```

The call to this Avenue Wrap is:

     Set thePoint = **avPointZMake**(xPt, yPt)

avPointZMake

GIVEN:	xPt	= X coordinate of point
	yPt	= Y coordinate of point
	zPt	= Z coordinate of point
RETURN:	thePoint	= the point feature

The given and returned variables should be declared where first called as:

```
Dim xPt As Double, yPt As Double, zPt As Double
Dim thePoint As IPoint
```

GEOMETRIC FEATURE CREATION

8.2.11 Function avPolygonMake

This function enables the programmer to create a polygon object from a given collection of points, which collection is composed as per Figure 8-2. The last point of said collection may or may not be a repetition of the first point. If it is not, the function will force a closure to the first point.

The corresponding Avenue request is:

```
thePolygon = Polygon.Make(shapeList)
```

avPolygonMake

The call to this Avenue Wrap is:

Set thePolygon = **avPolygonMake**(shapeList)

GIVEN: shapeList = the list of list of points comprising the polygon

RETURN: thePolygon = the polygon object feature

The given and returned variables should be declared where first called as:

```
Dim shapeList As New Collection
Dim thePolygon As IPolygon
```

8.2.12 Function avPolygonMake2

This function enables the programmer to create a polygon object from a given collection of points, which collection is composed as per Figure 8-1. The last point of said collection may or may not be a repetition of the first point. If it is not, the function will force a closure to the first point.

The corresponding Avenue request is:

There is no corresponding Avenue request.

avPolygonMake2

The call to this Avenue Wrap is:

Set thePolygon = **avPolygonMake2**(shapeList)

GIVEN: shapeList = the list of points comprising the polygon

RETURN: thePolygon = the polygon object feature

The given and returned variables should be declared where first called as:

```
Dim shapeList As New Collection
Dim thePolygon As IPolygon
```

GEOMETRIC FEATURE CREATION

8.2.13 Function avPolylineMake

This function enables the programmer to create a polyline object from a given collection of points, which collection is composed as per Figure 8-2.

The corresponding Avenue request is:

```
theLine = Polyline.Make(shapeList)
```

The call to this Avenue Wrap is:

avPolylineMake

Set theLine = **avPolylineMake**(shapeList)

GIVEN: shapeList = the list of points comprising the polygon

RETURN: theLine = the polyline feature

The given and returned variables should be declared where first called as:

```
Dim shapeList As New Collection
Dim theLine As IPolyline
```

8.2.14 Function avPolylineMake2

This function enables the programmer to create a polyline object from a given collection of points, which collection is composed as per Figure 8-1.

The corresponding Avenue request is:

There is no corresponding Avenue request.

The call to this Avenue Wrap is:

avPolylineMake2

Set theLine = **avPolylineMake2**(shapeList)

GIVEN: shapeList = the list of points comprising the polygon

RETURN: theLine = the polyline feature

The given and returned variables should be declared where first called as:

```
Dim shapeList As New Collection
Dim theLine As IPolyline
```

8.2.15 Function avPolyline2Pt

This function enables the programmer to create a polyline given the X and Y coordinates of two points.

The corresponding Avenue request is:

GEOMETRIC FEATURE CREATION

theLine = Polyline.Make({{X1 @ Y1, X2 @ Y2}})

The call to this Avenue Wrap is:

Set theLine = **avPolyline2Pt**(X1, Y1, X2, Y2)

avPolyline2Pt

GIVEN:	X1, Y1	= X and Y coordinate of the start point
	X2, Y2	= X and Y coordinate of the end point
RETURN:	theLine	= the polyline feature

The given and returned variables should be declared where first called as:

```
Dim X1, Y1, X2, Y2 As Double
Dim theLine As IPolyline
```

8.2.16 Function avRectMake4Pt

This function enables the programmer to create a rectangle given the X and Y coordinates for four corners which comprise the rectangle. The direction in which the corners are specified may be clockwise or counter-clockwise.

The corresponding Avenue request is:

There is no corresponding Avenue request.

The call to this Avenue Wrap is:

Set theRect = **avRectMake4Pt**(X1, Y1, X2, Y2, X3, Y3, X4, Y4)

avRectMake4Pt

GIVEN:	X1, Y1	= X and Y coordinate of corner point 1
	X2, Y2	= X and Y coordinate of corner point 2
	X3, Y3	= X and Y coordinate of corner point 3
	X4, Y4	= X and Y coordinate of corner point 4
RETURN:	theRect	= the rectangle (polygon) feature

The given and returned variables should be declared where first called as:

```
Dim X1, Y1, X2, Y2, X3, Y3, X4, Y4 As Double
Dim theRect As IPolygon
```

GEOMETRIC FEATURE CREATION

8.2.17 Function avRectMakeXY

This function enables the programmer to create a rectangle given the X and Y coordinates of two opposite corners.

The corresponding Avenue request is:

```
theRect = Rect.MakeXY(X1, Y1, X2, Y2)
```

The call to this Avenue Wrap is:

Set theRect = **avRectMakeXY**(X1, Y1, X2, Y2)

avRectMakeXY

GIVEN:	X1, Y1	= X and Y coordinate of the start point of a diagonal
	X2, Y2	= X and Y coordinate of the end point of a diagonal
RETURN:	theRect	= the rectangle (polygon) feature

The given and returned variables should be declared where first called as:

```
Dim X1, Y1, X2, Y2 As Double
Dim theRect As IPolygon
```

GEOMETRIC FEATURE CREATION

8.3 Geometric Attributes Avenue Wraps

GEOMETRIC ATTRIBUTES

8.3.1 Function avHasM

This function enables the programmer to determine if a given geometry object has an M attribute assigned to it. This function will handle point, multipoint, polyline, polygon and envelope objects.

The corresponding Avenue request is:
There is no corresponding Avenue request.

The call to this Avenue Wrap is:
hasM = **avHasM**(theGeom)

avHasM

GIVEN: theGeom = the geometry to be processed

RETURN: hasM = true if the geometry has an M value assigned to it, otherwise, false

The given and returned variables should be declared where first called as:
Dim theGeom As IGeometry
Dim hasM As Boolean

8.3.2 Function avHasZ

This function enables the programmer to determine if a given geometry object has a Z attribute assigned to it. This function will handle point, multipoint, polyline, polygon and envelope objects.

The corresponding Avenue request is:
hasZ = aShape.HasZ

The call to this Avenue Wrap is:
hasZ = **avHasZ**(theGeom)

avHasZ

GIVEN: theGeom = the geometry to be processed

RETURN: hasZ = true if the geometry has a Z value assigned to it, otherwise, false

The given and returned variables should be declared where first called as:
Dim theGeom As IGeometry
Dim hasZ As Boolean

GEOMETRIC ATTRIBUTES

8.3.3 Function avReturnArea

This function enables the programmer to get the area of an IGeometry object. Note that if an invalid geometry is specified, the function, avReturnArea, will return zero.

The corresponding Avenue request is:

```
     theArea = theGeom.ReturnArea
```

avReturnArea

The call to this Avenue Wrap is:

     theArea = **avReturnArea**(theGeom)

GIVEN:	theGeom	= the geometry to be processed
RETURN:	theArea	= the area of the geometry

The given and returned variables should be declared where first called as:

```
Dim theGeom As IGeometry
Dim theArea As Double
```

8.3.4 Function avReturnCenter

This function enables the programmer to get a point object representing the center of an IGeometry object. Note that if an invalid geometry is specified, the function, avReturnCenter, will return NOTHING.

The corresponding Avenue request is:

```
     theCenter = theGeom.ReturnCenter
```

avReturnCenter

The call to this Avenue Wrap is:

     Set theCenter = **avReturnCenter**(theGeom)

GIVEN:	theGeom	= the geometry to be processed
RETURN:	theCenter	= the centroid of the geometry

The given and returned variables should be declared where first called as:

```
Dim theGeom As IGeometry
Dim theCenter As IPoint
```

GEOMETRIC ATTRIBUTES

8.3.5 Function avReturnLength

This function enables the programmer to get the length of an IGeometry object (length of a line, perimeter of a polygon or circumference of a circle). When using this function, note the following:

1. For multi-part features, avReturnLength will return the total length, which includes all parts.
2. If an invalid geometry is specified the function, avReturnLength, will return zero.

The corresponding Avenue request is:

```
theLength = theGeom.ReturnLength
```

The call to this Avenue Wrap is:

theLength = **avReturnLength**(theGeom)

avReturnLength

GIVEN: theGeom = the geometry to be processed

RETURN: theLength = the length as described above

The given and returned variables should be declared where first called as:

```
Dim theGeom As IGeometry
Dim theLength As Double
```

GEOMETRIC ATTRIBUTES

8.4 Geometric Editing Avenue Wraps

GEOMETRIC EDITING

Reference is made to the functions avReturnIntersection, avReturnMerged and avReturnUnion presented below. The general operation of and differences between these three functions are identified below. In perusing them, refer to Figure 8-3(b).

- All three operate on a given pair of similar geometry feature types of multipoint, polyline or polygon.
- avReturnIntersection returns only those points, line segments or polygon areas that are common to both given features.
- avReturnMerged returns only those points, line segments or polygon areas that are not common to both given features.
- avReturnUnion returns all points, line segments or polygons except those that are duplicate to both given features.

8.4.1 Function avClean

This function enables the programmer to verify and enforce the correctness of a shape. In general, this means that duplicate points, vertices and line segments are removed from the shape.

The corresponding Avenue request is:

```
    CleanShape = aShape1.Clean
```

The call to this Avenue Wrap is:

```
    Set CleanShape = avClean(aShape1)
```

avClean

GIVEN: aShape1 = shape to be cleaned

RETURN: CleanShape = new shape reflecting the cleaning

The given and returned variables should be declared where first called as:

```
Dim aShape1 As IGeometry
Dim CleanShape As IGeometry
```

GEOMETRIC EDITING

8.4.2 Function avIntersects

This function enables the programmer to check whether two shapes intersect with each other.

The corresponding Avenue request is:

```
    anIntersect = aShape1.Intersects(aShape2)
```

The call to this Avenue Wrap is:

avIntersects

    anIntersect = **avIntersects**(aShape1, aShape2)

GIVEN:	aShape1	= base shape
	aShape2	= second shape intersected with the base shape
RETURN:	anIntersect	= intersection flag of the input objects. If: true = shapes intersect, false = they do not

The given and returned variables should be declared where first called as:

```
Dim aShape1 As IGeometry, aShape2 As IGeometry
Dim anIntersect As Boolean
```

8.4.3 Function avIsWithin

This function enables the programmer to determine if a shape (aShape1) is within a distance of another shape (aShape2).

The corresponding Avenue request is:

```
    isWithin = aShape1.IsWithin(aShape2, aDistance)
```

The call to this Avenue Wrap is:

avIsWithin

    isWithin = **avIsWithin**(aShape1, aShape2, aDistance)

GIVEN:	aShape1	= geometry object to be checked
	aShape2	= geometry object aShape1 is compared against
	aDistance	= distance value
RETURN:	isWithin	= flag denoting if aShape1 is close to aShape2. If: true = it is, false = it is not

The given and returned variables should be declared where first called as:

```
Dim aShape1 As IGeometry, aShape2 As IGeometry
Dim aDistance As Double
Dim isWithin As Boolean
```

GEOMETRIC EDITING

8.4.4 Function avReturnDifference

This function enables the programmer to subtract one shape from another to form a new shape. The portion which is subtracted from the base shape is the overlap with the second shape, see Figure 8-3(b). If there is no overlap, the shape that is passed back will be identical to the base shape.

The corresponding Avenue request is:

```
NewShape = aShape1.ReturnDifference(aShape2)
```

The call to this Avenue Wrap is:

```
Set NewShape = avReturnDifference(aShape1, aShape2)
```

avReturn Difference

GIVEN: aShape1 = base shape
aShape2 = second shape whose overlap with the base shape will be subtracted from the base shape.

RETURN: NewShape = new shape reflecting the difference, if any

The given and returned variables should be declared where first called as:

```
Dim aShape1 As IGeometry, aShape2 As IGeometry
Dim NewShape As IGeometry
```

8.4.5 Function avReturnIntersection

This function enables the programmer to intersect two shapes to form a new shape. If the shapes do not intersect the shape passed back, NewShape, will be an empty shape. When dealing with polygon shapes make sure the polygon is defined in a clockwise direction, if not, an intersection may not be computed. If two polylines are to be intersected, the resultant shape will be a point or multi-point shape (not the overlap of the two polylines).

The corresponding Avenue request is:

```
NewShape = aShape1.ReturnIntersection(aShape2)
```

The call to this Avenue Wrap is:

```
Set NewShape = avReturnIntersection(aShape1, aShape2)
```

avReturn Intersection

GIVEN: aShape1 = base shape
aShape2 = second shape to be intersected with the base shape

RETURN: NewShape = new shape reflecting the intersection, if any

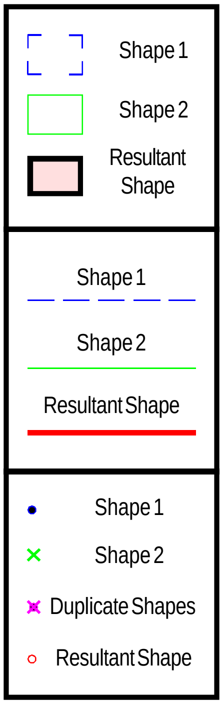
GEOMETRIC EDITING

The given and returned variables should be declared where first called as:

```
Dim aShape1 As IGeometry, aShape2 As IGeometry
Dim NewShape As IGeometry
```

8.4.6 Function avReturnMerged

This function enables the programmer to merge two shapes together to form a new shape. Refer to the commentary at the beginning of this section and to Figure 8-3(b) regarding the given shapes and returned shape of this Avenue Wrap.



Legend for Figure 8-3(b)

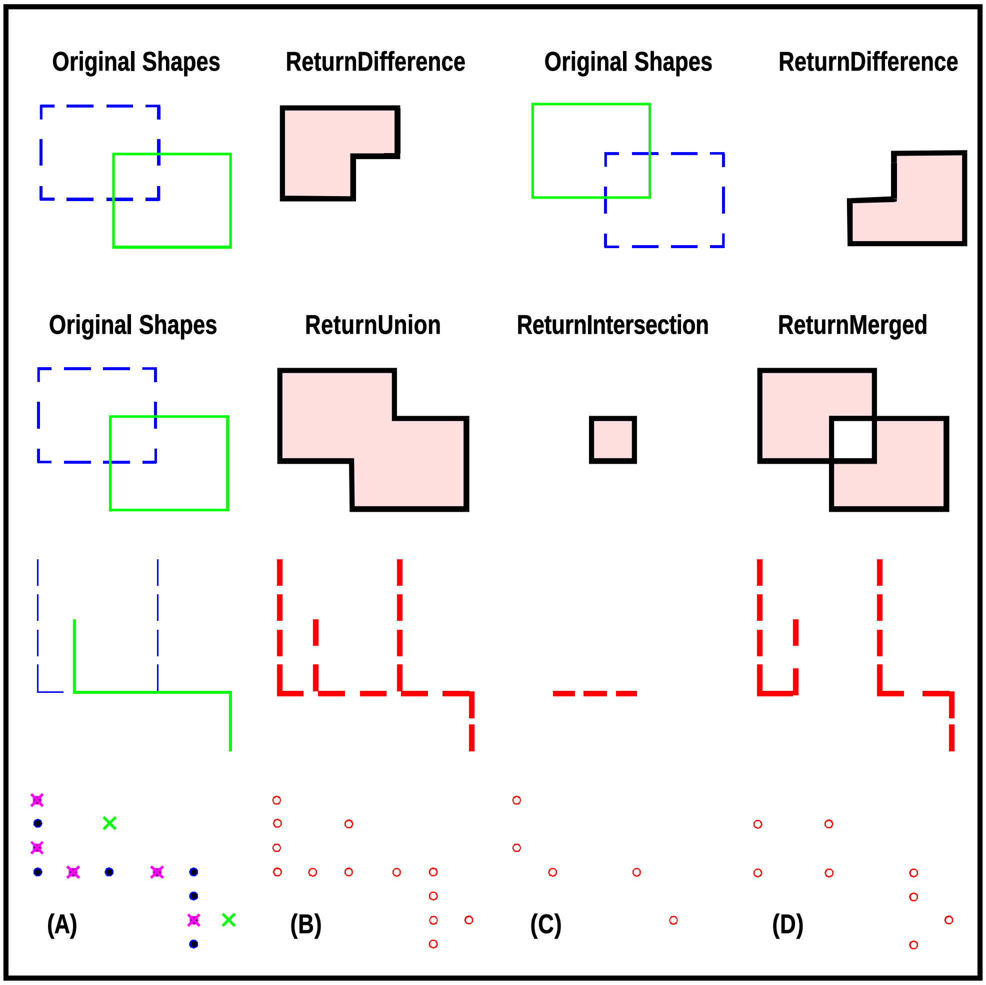


Figure 8-3(b) Difference, Union, Intersection and Merging of Two Shapes

GEOMETRIC EDITING

The corresponding Avenue request is:
NewShape = aShape1.ReturnMerged(aShape2)

The call to this Avenue Wrap is:
Set NewShape = **avReturnMerged**(aShape1, aShape2)

avReturnMerged

GIVEN: aShape1 = base shape
aShape2 = second shape merged with the base shape

RETURN: NewShape = new shape reflecting the merging

The given and returned variables should be declared where first called as:
Dim aShape1 As IGeometry, aShape2 As IGeometry
Dim NewShape As IGeometry

8.4.7 Function avReturnUnion

This function enables the programmer to union two shapes together to form a new shape. Refer to the commentary at the beginning of this section and to Figures 8-3(a) and 8-3(b) regarding the given shapes and returned shape of this Avenue Wrap.

The corresponding Avenue request is:
NewShape = aShape1.ReturnUnion(aShape2)

The call to this Avenue Wrap is:
Set NewShape = **avReturnUnion**(aShape1, aShape2)

avReturnUnion

GIVEN: aShape1 = base shape
aShape2 = second shape to be united with the base shape

RETURN: NewShape = new shape reflecting the union

The given and returned variables should be declared where first called as:
Dim aShape1 As IGeometry, aShape2 As IGeometry
Dim NewShape As IGeometry

8.4.8 Subroutine avSplit

This function enables the programmer to split a shape (line, polyline or polygon) using a second shape (a line or polyline) as a splitter. A collection of shapes is returned, which may be comprised of two or more shapes, depending on the configuration of the shape to be split and the splitter shape.

GEOMETRIC EDITING

The corresponding Avenue request is:

```
shapeList = aShape1.Split(aShape2)
```

The call to this Avenue Wrap is:

avSplit

Call **avSplit**(aShape1, aShape2, shapeList)

GIVEN:	aShape1	= shape to be split
	aShape2	= shape to be used as the split line
RETURN:	shapeList	= list of new shapes created as a result of the splitting process

The given and returned variables should be declared where first called as:

```
Dim aShape1 As IGeometry, aShape2 As IGeometry
Dim shapeList As New Collection
```

8.4.9 Subroutine avUnion

This subroutine enables the programmer to union two or more shapes to form a new shape. The programmer passes in a collection (list) of geometry objects. All objects in the given collection (list) must be of the same type. This procedure will handle point, multipoint, polyline and polygon objects.

The corresponding Avenue request is:

There is no corresponding Avenue request.

The call to this Avenue Wrap is:

avUnion

Call **avUnion**(pMap, geomList, pNewGeom)

GIVEN:	pMap	= map to be processed
	geomList	= collection of geometry objects to be unioned
RETURN:	pNewGeom	= new shape reflecting the unioning

The given and returned variables should be declared where first called as:

```
Dim pMap As IMap
Dim geomList As New Collection
Dim pNewGeom As IGeometry
```

8.5 Sample Code for Shape Editing

SAMPLE CODE

The example below demonstrates the use of the shape editing Avenue Wraps presented in the previous section. Four sample tests are carried out: (a) splitting of a polygon, (b) merging of two polygons, (c) intersection of two polygons, and (d) union of two polygons. To use the sample code below, do the following:

✍ **1** Create a module with the ArcMap VBA Editor and load or key enter the sample code below.

✍ **2** Go back to ArcMap and, using conventional ArcMap functionality, create seven polygons and and one polyline. In drawing these features, intersect the polyline with the first polygon that is drawn, and draw the other six polygons as three pairs of overlapping polygons.

✍ **3** In drawing the polygons and polyline, following the drawing order shown below:

(a) the polygon which is to be split by a polyline,
(b) the polyline to be used in splitting the polygon,
(c) the two polygons to be merged,
(d) the two polygons to be intersected, and
(e) the two polygons to be United.

✍ **4** Go back to the ArcMap Editor and execute the module with the sample code by clicking at the ▶ tool. If less then seven features were selected, a message will be displayed to this effect and the program will terminate, in which case go back to Step 3 above. If more than seven polygons are selected, only the first seven will be considered. The order of how the features are processed is based upon the order in which they were created. That is why the order of feature creation is important. Upon completion of each of the four tests, a message will be displayed and the resultant shape of the operation that was performed will be highlighted. At the end of the fourth pass, the program will terminate.

✍ **5** If desired, go back to Step 2 above, and repeat the test by modifying the figures that were drawn, and observe the results.

SAMPLE CODE

```
’
’  ---
’  ---Sample code illustrating how to perform various shape
’  ---editing operations.
’  ---This sample requires that seven polygon features
’  ---and one polyline feature be selected prior to
’  ---executing this macro.
’  ---The first selected polygon and the selected polyline
’  ---features will be used in a split operation.
’  ---The remaining selected polygons will be used to
’  ---demonstrate the merging, intersecting and uniting
’  ---operations.
’  ----
’
```

Declaration Statements

```
   Dim pMxApp As IMxApplication
   Dim pmxDoc As IMxDocument
   Dim pActiveView As IActiveView
   Dim pMap As IMap
   Dim selPG As ISelectionSet
   Dim selPL As ISelectionSet
   Dim selPGlist As New Collection
   Dim selPLlist As New Collection
   Dim iOpr As Integer
   Dim pFeatPG As IFeature
   Dim pFeatPL As IFeature
   Dim pGeomPG As IGeometry
   Dim pGeomPL As IGeometry
   Dim theOpr As String
   Dim pFeatPG1 As IFeature
   Dim pFeatPG2 As IFeature
   Dim pGeomPG1 As IGeometry
   Dim pGeomPG2 As IGeometry
   Dim shapeList As New Collection
   Dim mergedPoly As IGeometry
   Dim intrsPoly As IGeometry
   Dim unionPoly As IGeometry
   Dim i As Long
   Dim pg As IGeometry
   Dim pCurGraLyr1 As IGraphicsLayer
   Dim graPT As IElement
   Dim pSymbol As ISymbol
   Dim iIntrs As Boolean
’
’  ---Get the active view
```

Get the Document and the Polygon Selections

```
   Call avGetActiveDoc(pMxApp, pmxDoc, pActiveView, pMap)
’
’  ---Get the selected polygons from the theme L_0pg
   Call avGetSelection(pmxDoc, “L_0pg”, selPG)
```

SAMPLE CODE

```
'
'  ---Get the selected polyline from the theme L_0pl
   Call avGetSelection(pmxDoc, “L_0pl”, selPL)
'
'  ---Get the OIDs for the selection sets
   Call avGetSelectionIDs(selPG, selPGlist)
   Call avGetSelectionIDs(selPL, selPLlist)
'
'  ---Note that at least seven polygons and one polyline
'  ---must have been selected prior to invoking this
'  ---subroutine
   If ((selPGlist.Count > 6) And (selPLlist.Count > 0)) Then
'
'     ---Perform 4 shape editing operations
'     ---Loop 1 splits a polygon (selected polygon #1) using
'     ---the selected polyline as a splitter.
'     ---Loop 2 merges two polygons (selected polygons #2 and
'     ---#3)
'     ---Loop 3 intersects two polygons (selected polygons #4
'     ---and #5)
'     ---Loop 4 unites two polygons (selected polygons #6 and
'     ---#7)
'
      For iOpr = 1 To 4
'
'         ---Get the first selected polygon and the polyline
          If (iOpr = 1) Then
             Call avGetFeature(pmxDoc, “L_0pg”, _
                                  selPGlist.Item(1), pFeatPG)
             Call avGetFeature(pmxDoc, “L_0pl”, _
                                  selPLlist.Item(1), pFeatPL)
'            ---Get the geometries of the selected features
             Set pGeomPG = pFeatPG.Shape
             Set pGeomPL = pFeatPL.Shape
'            ---Define the editing operation
             theOpr = “Split”
'
'         ---The other editing operations require two
'         ---polygons, so get the next selected polygons
          Else
             If (iOpr = 2) Then
                Call avGetFeature(pmxDoc, “L_0pg”, _
                                    selPGlist.Item(2), pFeatPG1)
                Call avGetFeature(pmxDoc, “L_0pg”, _
                                    selPGlist.Item(3), pFeatPG2)
'               ---Define the editing operation
                theOpr = “Merge”
             End If
```

Get the OIDs associated with the selections

Get the Features from the Selected Collections for Splitting

Get the Features from the Selected Collections for Merging

SAMPLE CODE

Get the Features from the Selected Collections for Intersecting

```
             If (iOpr = 3) Then
                Call avGetFeature(pmxDoc, “L_0pg”, _
                                 selPGlist.Item(4), pFeatPG1)
                Call avGetFeature(pmxDoc, “L_0pg”, _
                                 selPGlist.Item(5), pFeatPG2)
'               ---Define the editing operation
                theOpr = “Intersect”
             End If
```

Get the Features from the Selected Collections for Unioning

```
             If (iOpr = 4) Then
                Call avGetFeature(pmxDoc, “L_0pg”, _
                                 selPGlist.Item(6), pFeatPG1)
                Call avGetFeature(pmxDoc, “L_0pg”, _
                                 selPGlist.Item(7), pFeatPG2)
'               ---Define the editing operation
                theOpr = “Union”
             End If
'            ---Get the geometries of the selected features
             Set pGeomPG1 = pFeatPG1.Shape
             Set pGeomPG2 = pFeatPG2.Shape
          End If
'
'         ---Split the polygon using the polyline
          If (iOpr = 1) Then
```

Loop 1
Split a Polygon

```
             Call avSplit(pGeomPG, pGeomPL, shapeList)
          End If
'
'         ---Merge two polygons together (will create a
'         ---hole if the polygons overlap)
          If (iOpr = 2) Then
```

Loop 2
Merge Two Polygons

```
             Set mergedPoly = avReturnMerged(pGeomPG1, _
                                             pGeomPG2)
             Call CreateList(shapeList)
             shapeList.Add mergedPoly
          End If
'
'         ---Intersect two polygons (returns an empty shape
'         ---if the polygons do not intersect)
          If (iOpr = 3) Then
```

Loop 3
Intersect Two Polygons

```
             Set intrsPoly = avReturnIntersection(pGeomPG1, _
                                                  pGeomPG2)
             Call CreateList(shapeList)
             shapeList.Add intrsPoly
          End If
'
'         ---Union two polygons
          If (iOpr = 4) Then
```

Loop 4
Union Two Polygons

```
             Set unionPoly = avReturnUnion(pGeomPG1, _
                                           pGeomPG2)
```

SAMPLE CODE

```
             Call CreateList(shapeList)
             shapeList.Add unionPoly
          End If
'
'         ---Check if any new polygons were created.  If so
'         ---cycle thru them, and display each new polygon
'         ---in red to indicate the shape of the new polygon.
'         ---In addition, display in a message box the area
'         ---of the polygon and the operation that was
'         ---performed.
          If (shapeList.Count > 0) Then
             For i = 1 To shapeList.Count
'                ---Grab a polygon from the list
                 Set pg = shapeList.Item(i)
'                ---Set the current active graphics layers as
'                ---the basic graphics layer
                 Call avSetGraphicsLayer(Null, pCurGraLyr1)
'                ---Create a graphic polygon using the new
'                ---polygon as its shape and assign a red
'                ---fill to it
                 Set graPT = avGraphicShapeMake(“FILL”, pg)
                 Set pSymbol = avSymbolMake(“FILL”)
                 Call avSymbolSetColor(“FILL”, pSymbol, _
                                       “RED”)
                 Call avGraphicSetSymbol(“FILL”, graPT, _
                                         pSymbol)
'                ---Add the graphic to the display
                 Call avViewAddGraphic(graPT)
'                ---Display in a message box the area of the
'                ---new polygon and the operation that was
'                ---performed
                 If (iOpr <> 3) Then
                    MsgBox theOpr + “ operation” + Chr(13) + _
                         “Polygon “ + CStr(i) + “ Area = “ + _
                         CStr(avReturnArea(pg))
                 Else
'                   ---Determine if the polygons intersect
                    iIntrs = avIntersects(pGeomPG1, pGeomPG2)
'                   ---In addition to the usual information,
'                   ---inform user whether the polygons
'                   ---intersect each other or not
                    MsgBox theOpr + “ operation” + Chr(13) + _
                        “Polygon “ + CStr(i) + “ Area = “ + _
                        CStr(avReturnArea(pg)) + Chr(13) + _
                        “Intersection = “ + CStr(iIntrs)
                 End If
'                ---Get rid of the graphic
                 Call avRemoveGraphic(graPT)
             Next
```

Inform user as to the results of the editing process

SAMPLE CODE

```
'         ---Handle the case when and operation does not
'         ---produce new polygons
          Else
             MsgBox theOpr + “ produced no new shapes”
          End If
      Next
'
'  ---Handle the case when not enough features selected for
'  ---the various editing operations to be performed
   Else
      MsgBox “Not enough features selected”
   End If
'
```