Geometries

Content

1. Geometries. introduction

This interface is equivalent to GM_Object specified in ISO 19107. It is the root class of the geometric object taxonomy and supports common interfaces to all geographically referenced geometric objects.

Geometry are direct positions in a coordinate systems of a particular reference. A geometry can be considered as an infinite set of points, that satisfies the setting operation interfaces of a direct set of positions.

A geometric object is a combination of a coordinate geometry and a system of coordinates. For all operations, all geometric calculations are done in the reference coordinate system of the first geometric object accessed, which usually is the object whose operation is being invoked. Items returned must be in the reference coordinate system in which the calculations are made, unless there is there is another specification.

2.The concept of type and subtype

In gvSIG, type only refers to the object geometry class depending on their characteristics, but does not indicate if the geometry is 2-dimensional or 3-dimensional, or if it supports a coordinate M or if it does not support it. For this reason, we have developed the concept of a subtype used to indicate the dimension of a geometry.

The geometric types currently supported by gvSIG are defined in TYPES, those are :

- Simple geometries
  - AGGREGATE: A set that can contain points, lines and polygons
  - LINE | CURVE: A continuous line or curve geometry, with indefinite amount of points. The constant can be used interchangeably or CURVE LINE, represents the same type of geometry.
  - POINT: Dimensionless geometric figure: it has no length, area, volume, or other dimensional angle. Describes a position in space, determined on a preset coordinate system.
  - POLYGON | SURFACE: Plane figure composed of a finite sequence of straight line segments enclosing a region in space. The constant can be used interchangeably or in a CURVE LINE, it represents the same type of geometry.
- Multiple geometries:
  - MULTICURVE: Set of lines (either curved or straight)
  - MULTIPOINT: Set of points
  - MULTISURFACE: Set of surfaces

The gvSIG subtypes are defined in SUBTYPES, witch are:

- D2 (GEOM2D): 2D geometries
- D2M (GEOM2DM): 2-dimensional geometries and M coordinates
- D3 (GEOM3D): 3-dimensional geometries
- D3M (GEOM3DM): : 3-dimensional geometries and M coordinates

3. Creating geometries

Within gvSIG scripting environment, it's been created a module called geom.py, which simplifies some of the most common operations for geometries.

To create a geometry in our script we must import the module 'geom' and invoke *createGeometry* function indicating the type and subtype of the new geometry.

```
from geom import *


def main():
  geometry = createGeometry(POINT, D2)
```

The geometry then created is a *point* type geometry, subtype *2d* empty.

4. Geometry type Point. Function createPoint

We can create a point-type geometry (POINT) by geometric module *createPoint* function that returns a geometry and whose syntax is:

```
createPoint([x, y][, subtype])
```

- createPoint(x, y, subtype)
  - x, double (opcional): Value of the point in X coordinate. The default value is 0.0
  - y, double (opcional): Valor of the point in Y coordinate. The default value is 0.0
  - subtype, int (optional): Specifies the size of the geometry. It supports z coordinate and the default value is D2

Common methods of point-type geometry are:

- getCoordinateAt (dimension): compilment the coordinate value in a particular dimension.
  - dimension, int: dimension to obtain the coordinate
- getX(): Returns the X coordinate value
- getY(): Returns the Y coordinate value
- setCoordinateAt (dimension, value): Sets the value of a particular dimension
  - dimension, int: Dimension establishing the coordinate
  - value, double: value to assign to the coordinate
- setX(x): Sets the value of the X coordinate
  - x, double: value assigned to the X coordinate
- setY(y): Sets the value of the Y coordinate
  - y, double: alue assigned to the Y coordinate

Suppose we have a point geometry type and we want to get the coordinates "x" and "y" in order to create another point type geometry with those values.

```
from geom import *


def main():
  #
```

```
 #code ...
x = point.getX()
y = point.getY()


newPoint = createPoint(x,y)
```

To edit or retrieve other values of other dimensions, we have to use generic methods; *getCoordinateAt* and *setCoordinateAt.*

For example, to create a 3-dimensional point whose coordinates are 1,1,5 we have to execute the following code:

```
point = createGeometry(POINT, D3)
point.setCoordinateAt(0, 1)
point.setCoordinateAt(1, 1)
point.setCoordinateAt(2, 5)
```

Another option could be done using the methods setX () and setY () of the geometry and the method setCoordinateAt (dimension, value) and the point obtained will be the same:

```
point = createGeometry(POINT, D3)
point.setX(1)
point.setY(1)
point.setCoordinateAt(2, 5)
```

For more complete information on methods of point-type geometry, check the documentation; type geometry Point.

5. Polygon geometries. Function createPolygon

CreatePolygon function from the *geom* module allows us to create a polygon type geometry (POLYGON) in a quickly method.

The syntax of the function is

```
createPolygon([subtype])
```

- createPolygon(subtype): Creates a polygon type geometry.
    - subtype, int (optional): Specifies the size of the geometry and tells if it supports a z coordinate. The default value is D2

Common methods of these geometries are:

- addVertex(point): Adds a vertex to the geometry
    - point, Point: Point geometry type to add to the polygon
- closePrimitive(): Closes the geometry
- getBounds(): Returns the Rectangle rectangular geometry limit.
- getCoordinateAt(int index, int dimension): Gets the value of the coordinate with a particular dimension
- getEnvelope(): Returns the minimum bounding box of the geometry
- getNumVertices(): Returns the number of vertices of the geometry
- getVertex(int index): Returns a geometry vertex point *index* type of the geometry

For example, the following code creates a polygon type geometry, which adds 4 vertices.

```
from geom import *

    def main():
    polygon = createPolygon(D2M)

    x = (1, 1, 5, 5)
    y = (1, 5, 5, 1)
    points = [createPoint(p[0], p[1]) for p in zip(x, y)]
    for point in points:
        polygon.addVertex(point)

    polygon.closePrimitive()
```

6. Geometry type Line. Function createLine

CreateLine function from *geom* module that allows us to create a line geometry type quickly.

The syntax of the function is

```
createLine([subtype])
```

- createLine(subtype): Creates a line geometry type (LINE).
    - subtype, int (optional): Specifies the size of the geometry and if it supports a z coordinate. The default value is D2

Common methods of these geometries are:

- addVertex(point): Adds a vertex to the geometry
    - point, Point: Point geometryto add to the polygon
- closePrimitive(): Closes the geometry
- getBounds(): Returns the Rectangle rectangular geometry limit.
- getCoordinateAt(int index, int dimension): Gets the coordinate value of a particular dimension
- getEnvelope(): Returns the minimum bounding box of the geometry
- getNumVertices(): Returns the number of vertices of the geometry
- getVertex(int index): Returns a point geometry type of the vertex index of the geometries

7. Multipoint Geometry type. Function createMultiPoint

MULTIPOINT geometries are geometries which in contain in the smae time a collection of geometries of type POINT.

We can create a type geometry MULTIPOINT invoking the function *createMultiPoint*, indicating the subtype of geometry we want to create.

```
createMultiPoint([subtype])
```

- createMultiPoint(subtype): Creates a type geometry MULTIPOINT
  - subtype, int: geometry Subtype

The following code creates a MULTIPOINT type geometry and adds 6 geometries of type POINT

```
from geom import *

def main():
  multipoint = createMultiPoint(subtype=D2)
  x = (1, 1, 5, 5, 10, 0)
  y = (1, 5, 5, 1, 10, 10)
  points = [createPoint(p[0], p[1]) for p in zip(x, y)]
  for point in points:
    multipoint.addPoint(point)
```

We can also add points to a geometry MULTIPOINT going trough parameters as lists, one with the X and Y coordinates, and other with the points of the Y coordinates that will form the multi-geometry.

```
from geom import *

def main():
  multipoint = createMultiPoint(subtype=D2)
  x = (1, 1, 5, 5, 10, 0)
  y = (1, 5, 5, 1, 10, 10)
  multipoint.setPoints(x, y)
```

The MULTIPOINT class methods are:

```
- addPoint(point)
- cloneGeometry()
- getEnvelope()
- getPrimitivesNumber()
- setPoints(xCoords, yCoords)
- getPointAt(index)
```

- addPoint(Point point): Adds geometry to a multi-geometry type POINT
  - Point point: point to add in a geometry
- cloneGeometry(): Creates a copy of this geometry.
- getEnvelope(): Returns the minimum envelope of the geometry.
- getPrimitivesNumber(): Returns the number of geometries forming the multi-geometry
- setPoints(double[] xCoords, double[] yCoords): Inserts on the multi-geometry the corresponding points.
  - double[] xCoords: List of X coordinates of the points
  - double[] yCoords: List of Y coordinates of the points
- getPointAt(int index): returns the points of a particular position in the multi-geometry

- int index: position of the point

8. Multiline geometry type. Function createMultiLine

MULTILINE geometries are geometries which contain a collection of geometries of type LINE.

We can create a type geometry MULTILINE, invoking the function *createMultiLine*, indicating the subtype of geometry we want to create.

```
createMultiLine([subtype])
```

- createMultiLine(subtype): Creates a type geometry MULTILINE
  - subtype, int: geometry Subtype

The MULTILINE class methods are:

```
- addCurve(line)
- getCurveAt(index)
- cloneGeometry()
- getEnvelope()
- getPrimitivesNumber()
```

- addCurve(line): Adds a new line (LINE) to the multi-geometry
  - LINE line: A line that is added
- getCurveAt(index): Returns a line of a particular position in the multi-geometry
  - int index: line position within the multi-geometry
- cloneGeometry(): Creates a copy of this geometry.
- getEnvelope(): Return the minimum envelope of the geometry.
- getPrimitivesNumber(): Returns the number of geometries that form the multi-geometry

9.Geometry Multipolygon type. Function createMultiPolygon

MULTIPOLYGON geometries are geometries which contain a collection of type POLYGON geometries.

We can create a type geometry MULTIPOLYGON invoking the function *createMultiPolygon,* indicating the subtype of geometry we want to create.

```
createMultiPolygon([subtype])
```

- createMultiPolygon(subtype): Creates a type geometry MULTILINE
  - subtype, int geometry Subtype

The MULTIPOLYGON class methods are:

```
- addSurface(polygon)
- cloneGeometry()
- getSurfaceAt(index)
- cloneGeometry()
- getEnvelope()
```

- `getPrimitivesNumber()`

- addCurve(line): Adds a new line (LINE) to the multi-geometry
    - LINE line: A line that is added
- getCurveAt(index): returns a LINE of a particular position in the multi-geometry
    - line position within the multi-geometry
- cloneGeometry (): Creates a copy of this geometry.
- getEnvelope(): Returns the minimum envelope of the geometry.
- getPrimitivesNumber(): Returns the number of geometries that take part in a multi-geometry

10. Transactions with geometries

- area(): Returns the geometry area
- centroid(): Returns the centroid of the geometry
- perimeter(): Returns the perimeter of the geometry
- getEnvelope(): Returns the minimum bounding box of the geometry
- buffer(distance): Returns the resulting geometry after calculate an area around the geometry, by clicking the distance parameter .
    - distance, double: buffer radius.
- convexHull(): Returns the convex hull of the geometry
- distance(geometry): Returns the minimum distance between the geometry and the geometry of the parameter.
    - geometry, Geometry: Geometry distant
- intersection(other): Returns the resulting geometry of the intersection
    - other, Geometry: Geometry after the intersection
- intersects(geometry): Returns True if both geometries intersect
    - geometry, Geometry: Geometry-check
- closestPoints(geometry): Returns the closest points of the respective geometries
    - geometry[], Geometry: Points (POINT) closest to each of the geometries

For more information, see the API gvSIG geometries API geometries of gvSIG.

gvSIG Association

Plaza Don Juan de Villarrasa 14-5,

46001, Valencia (Spain)

Registro Nacional de Asociaciones (National Registration of Associations): 596206

e-mail : info@gvsig.com

Web: www.gvsig.com

Project website: http://www.gvsig.org

Documentation made by Víctor Acevedo. Translated by Elisabet Adeva

Distribution Lists

There are three mailing lists in order to facilitate communication between all stakeholders in gvSIG project. The first two are for users and developers, mainly oriented to the Hispanic community, with the use of the Castilian lenguage preferred. The third list, is a international list, aimed at other communities where is preferd the use of English.

- **Members List.** . Here you can leave your opinions on the operation of the software: what things would you like to be developed, doubts related to the use of gvSIG and anything you believe should be in discussion and accommodates in the list of users. The link to subscribe to the user list is the following:
  http://listserv.gva.es/mailman/listinfo/gvsig_usuarios

- **Developers list.** It is geared/oriented to anyone interested in knowing how gvSIG is developed. The link to subscribe to this list is:
  http://listserv.gva.es/mailman/listinfo/gvsig_desarrolladores

- **International list.** It is intended for both users and developers witch do not speak Spanish. The preferably language used is English. The link to subscribe to this list is:
  http://listserv.gva.es/mailman/listinfo/gvsig_internacional