Geoscript Groovy Cookbook

Jared Erickson

Table of Contents

Geometry Recipes

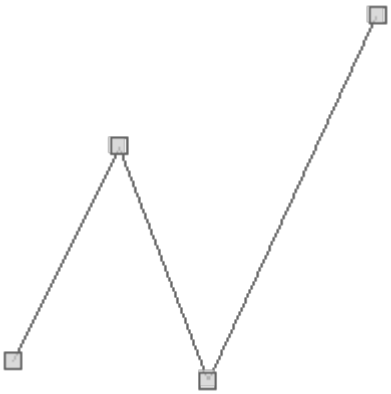
Creating Geometries

Create a Point with an XY

```
Point point = new Point(-123,46)
```

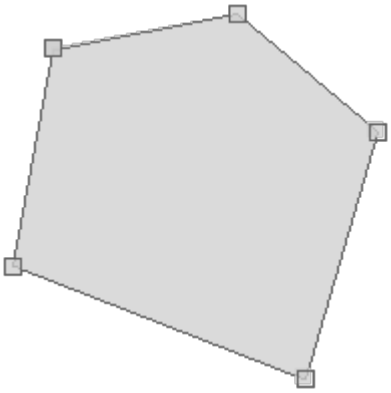
Create a LineString from Coordinates

```
LineString lineString = new LineString(
        [3.1982421875, 43.1640625],
        [6.7138671875, 49.755859375],
        [9.7021484375, 42.5927734375],
        [15.3271484375, 53.798828125]
)
```



Create a Polygon from a List of Coordinates

```
Polygon polygon = new Polygon([[
       [-101.35986328125, 47.754097979680026],
       [-101.5576171875, 46.93526088057719],
       [-100.12939453125, 46.51351558059737],
       [-99.77783203125, 47.44294999517949],
       [-100.45898437499999, 47.88688085106901],
       [-101.35986328125, 47.754097979680026]
]])
```

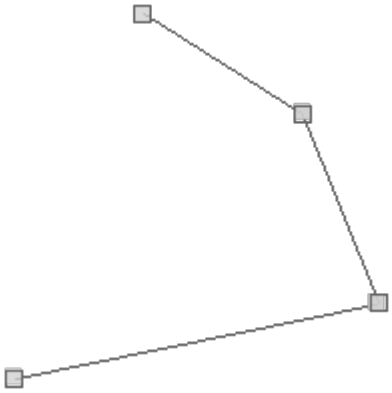


Create a MultiPoint with a List of Points

```
MultiPoint multiPoint = new MultiPoint([
        new Point(-122.3876953125, 47.5820839916191),
        new Point(-122.464599609375, 47.25686404408872),
        new Point(-122.48382568359374, 47.431803338643334)
])
```

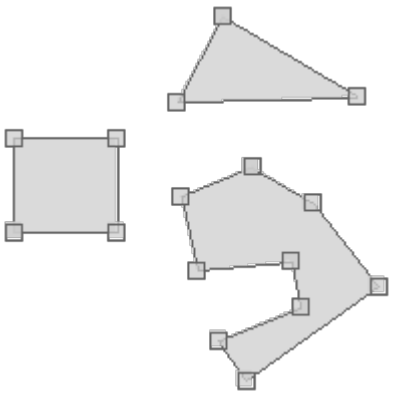
Create a MultiLineString with a List of LineStrings

```
MultiLineString multiLineString = new MultiLineString([
        new LineString (
                [-122.3822021484375, 47.57837853860192],
                [-122.32452392578125, 47.48380086737799]
        ),
        new LineString (
                [-122.32452392578125, 47.48380086737799],
                [-122.29705810546874, 47.303447043862626]
        ),
        new LineString (
                [-122.29705810546874, 47.303447043862626],
                [-122.42889404296875, 47.23262467463881]
        )
])
```



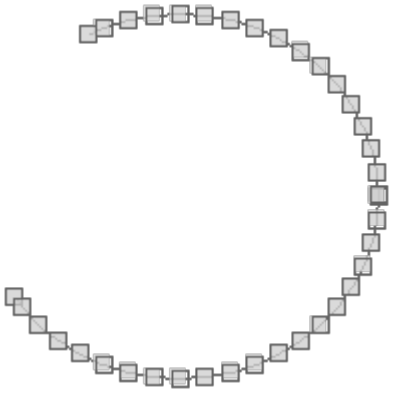
Create a MultiPolygon with a List of Polygons

```
MultiPolygon multiPolygon = new MultiPolygon(
    new Polygon ([[
            [-122.2723388671875,  47.818687628247105],
            [-122.37945556640624, 47.66168780332917],
            [-121.95373535156249, 47.67093619422418],
            [-122.2723388671875,  47.818687628247105]
    ]]),
    new Polygon ([[
            [-122.76672363281249, 47.42437092240516],
            [-122.76672363281249, 47.59505101193038],
            [-122.52227783203125, 47.59505101193038],
            [-122.52227783203125, 47.42437092240516],
            [-122.76672363281249, 47.42437092240516]
    ]]),
    new Polygon ([[
            [-122.20367431640624, 47.543163654317304],
            [-122.3712158203125,  47.489368981370724],
            [-122.33276367187499, 47.35371061951363],
            [-122.11029052734374, 47.3704545156932],
            [-122.08831787109375, 47.286681888764214],
            [-122.28332519531249, 47.2270293988673],
            [-122.2174072265625,  47.154237057576594],
            [-121.904296875,      47.32579231609051],
            [-122.06085205078125, 47.478232163128851],
            [-122.20367431640624, 47.543163654317304]
    ]])
)
```



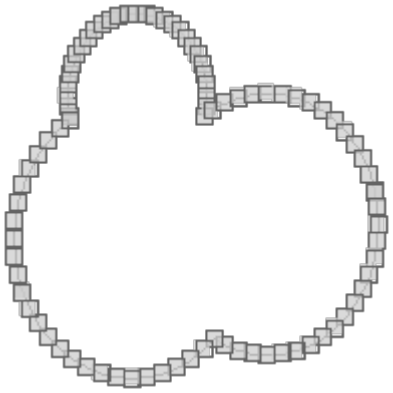
Create a CircularString with a List of Points

```
CircularString circularString = new CircularString([
        [-122.464599609375, 47.247542522268006],
        [-122.03613281249999, 47.37789454155521],
        [-122.37670898437499, 47.58393661978134]
])
```



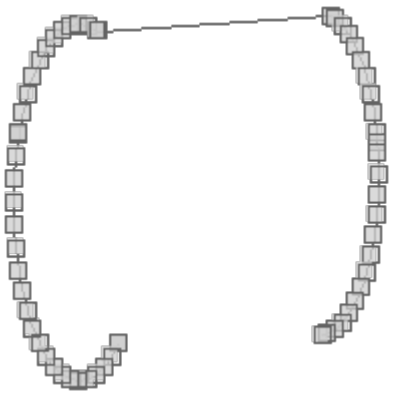
Create a CircularRing with a List of Points

```
CircularRing circularRing = new CircularRing([
        [-118.47656249999999, 41.508577297439324],
        [-109.6875, 57.51582286553883],
        [-93.8671875, 42.032974332441405],
        [-62.57812500000001, 30.14512718337613],
        [-92.10937499999999, 7.36246686553575],
        [ -127.265625, 14.604847155053898],
        [-118.47656249999999, 41.508577297439324]
])
```



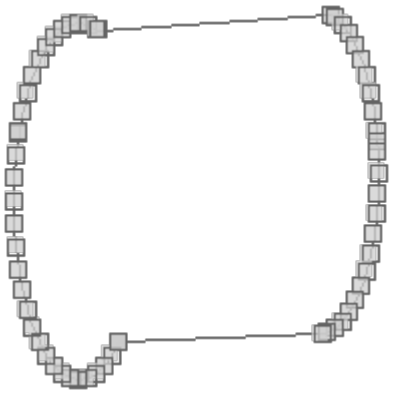
Create a CompoundCurve with a List of CircularStrings and LineStrings

```
CompoundCurve compoundCurve = new CompoundCurve([
    new CircularString([
            [27.0703125, 23.885837699862005],
            [5.9765625, 40.17887331434696],
            [22.5, 47.98992166741417],
    ]),
    new LineString([
            [22.5, 47.98992166741417],
            [71.71875, 49.15296965617039],
    ]),
    new CircularString([
            [71.71875, 49.15296965617039],
            [81.5625, 39.36827914916011],
            [69.9609375, 24.5271348225978]

    ])
])
```



Create a CompoundRing with a connected List of CircularStrings and LineStrings

```
CompoundRing compoundRing = new CompoundRing([
        new CircularString([
                [27.0703125, 23.885837699862005],
                [5.9765625, 40.17887331434696],
                [22.5, 47.98992166741417],
        ]),
        new LineString([
                [22.5, 47.98992166741417],
                [71.71875, 49.15296965617039],
        ]),
        new CircularString([
                [71.71875, 49.15296965617039],
                [81.5625, 39.36827914916011],
                [69.9609375, 24.5271348225978]

        ]),
        new LineString([
                [69.9609375, 24.5271348225978],
                [27.0703125, 23.885837699862005],
        ])
])
```



Procesing Geometries

Get the area of a Geometry

```
Polygon polygon = new Polygon([[
    [-124.80, 48.92],
    [-126.21, 45.33],
    [-114.60, 45.08],
    [-115.31, 51.17],
    [-121.99, 52.05],
    [-124.80, 48.92]
]])
double area = polygon.area
println area
```

```
62.4026
```

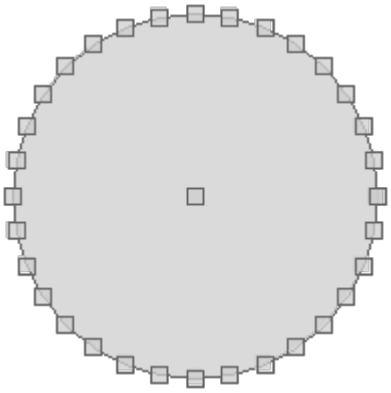
Get the length of a Geometry

```
LineString lineString = new LineString([-122.69, 49.61], [-99.84, 45.33])
double length = lineString.length
println length
```

```
23.24738479915536
```

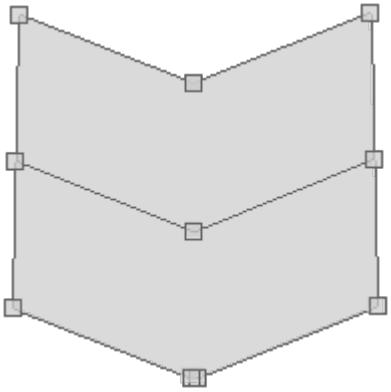
Buffer a Point

```
Point point = new Point(-123,46)
Geometry bufferedPoint = point.buffer(2)
```



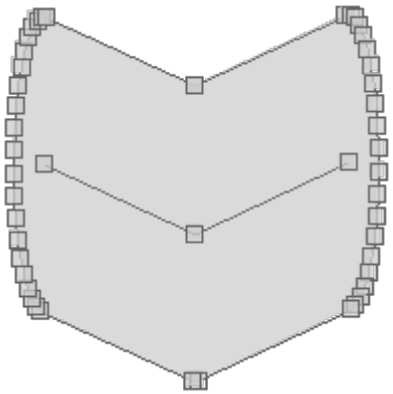
Buffer a LineString with a butt cap

```
LineString line = new LineString([
    [-122.563, 47.576],
    [-112.0166, 46.589],
    [-101.337, 47.606]
])
Geometry bufferedLine1 = line.buffer(2.1, 10, Geometry.CAP_BUTT)
```



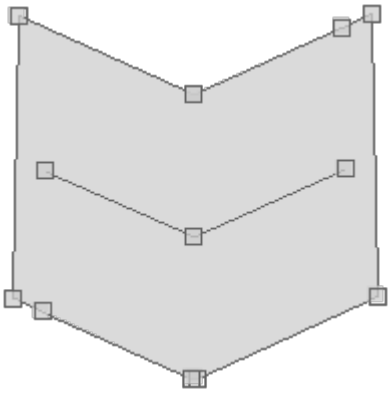
Buffer a LineString with a round cap

```
Geometry bufferedLine2 = line.buffer(2.1, 10, Geometry.CAP_ROUND)
```



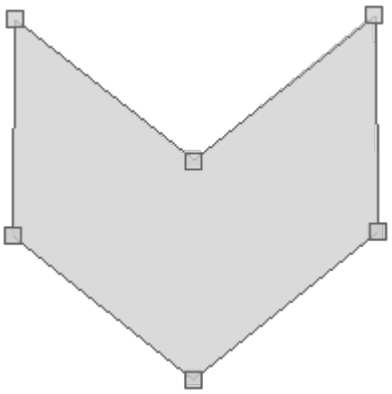
Buffer a LineString with a square cap

```
Geometry bufferedLine3 = line.buffer(2.1, 10, Geometry.CAP_SQUARE)
```



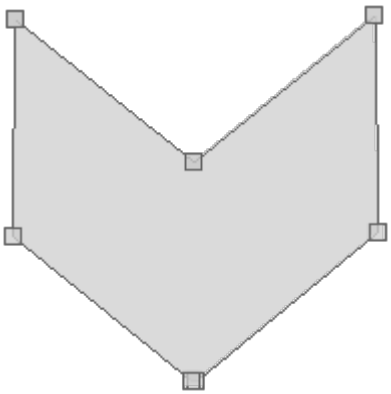
Buffer a LineString on the right side only

```
LineString line = new LineString([
        [-122.563, 47.576],
        [-112.0166, 46.589],
        [-101.337, 47.606]
])
Geometry rightBufferedLine = line.singleSidedBuffer(1.5)
```



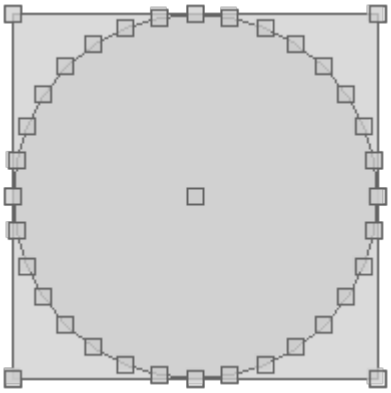
Buffer a LineString on the left side only

```
Geometry leftBufferedLine = line.singleSidedBuffer(-1.5)
```



Get Bounds from a Geometry

```
Point point = new Point(-123,46)
Polygon polygon = point.buffer(2)
Bounds bounds = polygon.bounds
```

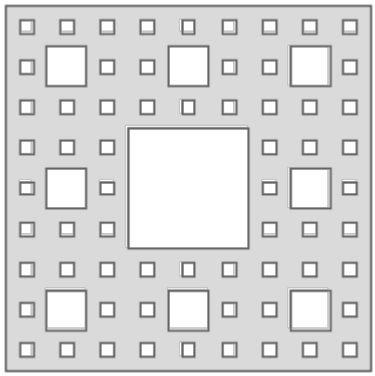


Create a Geometry of a String

```
Geometry geometry = Geometry.createFromText("Geo")
```

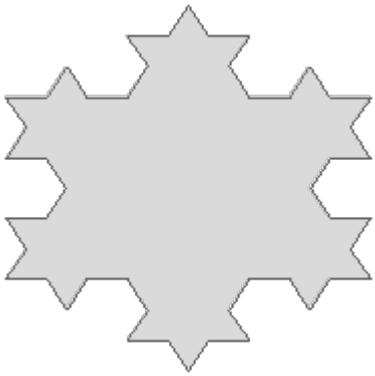
Create a Sierpinski Carpet in a given Bounds and with a number of points

```
Bounds bounds = new Bounds(21.645,36.957,21.676,36.970, "EPSG:4326")
Geometry geometry = Geometry.createSierpinskiCarpet(bounds, 50)
```



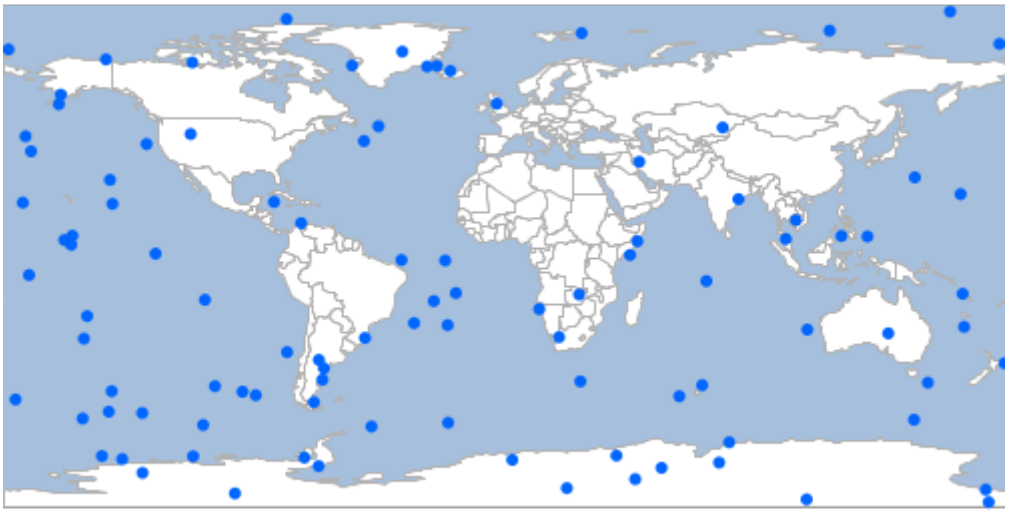
Create a Kock Snowflake in a given Bounds and with a number of points

```
Bounds bounds = new Bounds(21.645,36.957,21.676,36.970, "EPSG:4326")
Geometry geometry = Geometry.createKochSnowflake(bounds, 50)
```



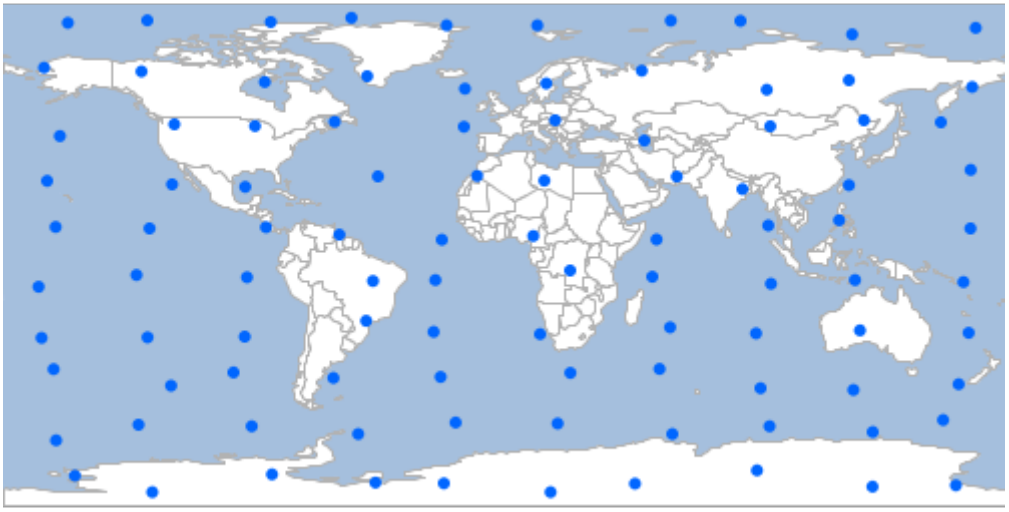
Create a number of random points within a given Geometry

```
Geometry geometry = new Bounds(-180, -90, 180, 90).geometry
MultiPoint randomPoints = Geometry.createRandomPoints(geometry, 100)
```



Create a number of random points within a given Geometry where the points are contrained to the cells of a grid

```
Bounds bounds = new Bounds(-180, -90, 180, 90)
MultiPoint randomPoints = Geometry.createRandomPointsInGrid(bounds, 100, true, 0.5)
```



Reading and Writing Geometries

The geoscript.geom.io package has several Readers and Writers for converting geoscript.geom.Geometry to and from strings.

Readers and Writers

Find all Geometry Readers

```
List<Reader> readers = Readers.list()
readers.each { Reader reader ->
    println reader.class.simpleName
}
```

```
GeobufReader
GeoJSONReader
GeoRSSReader
Gml2Reader
Gml3Reader
GpxReader
KmlReader
WkbReader
WktReader
GooglePolylineEncoder
```

Find a Geometry Reader

```
String wkt = "POINT (-123.15 46.237)"
Reader reader = Readers.find("wkt")
Geometry geometry = reader.read(wkt)
```

Find all Geometry Writers

```
List<Writer> writers = Writers.list()
writers.each { Writer writer ->
    println writer.class.simpleName
}
```

```
GeobufWriter
GeoJSONWriter
GeoRSSWriter
Gml2Writer
Gml3Writer
GpxWriter
KmlWriter
WkbWriter
WktWriter
GooglePolylineEncoder
```

Find a Geometry Writer

```
Geometry geometry = new Point(-122.45, 43.21)
Writer writer = Writers.find("geojson")
String geojson = writer.write(geometry)
println geojson
```

```
{"type":"Point","coordinates":[-122.45,43.21]}
```

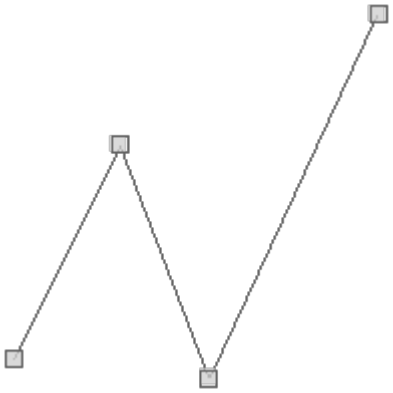
WKT

Read a Geometry from WKT using the WktReader

```
String wkt = "POINT (-123.15 46.237)"
WktReader reader = new WktReader()
Geometry geometry = reader.read(wkt)
```

Read a Geometry from WKT using the Geometry.fromWKT() static method

```
String wkt = "LINESTRING (3.198 43.164, 6.7138 49.755, 9.702 42.592, 15.327 53.798)"
Geometry geometry = Geometry.fromWKT(wkt)
```



Get the WKT of a Geometry

```
Geometry geometry = new Point(-123.15, 46.237)
String wkt = geometry.wkt
println wkt
```

```
POINT (-123.15 46.237)
```

Write a Geometry to WKT using the WktWriter

```
Geometry geometry = new LineString(
        [3.198, 43.164],
        [6.713, 49.755],
        [9.702, 42.592],
        [15.32, 53.798]
)
WktWriter writer = new WktWriter()
String wkt = writer.write(geometry)
println wkt
```

```
LINESTRING (3.198 43.164, 6.713 49.755, 9.702 42.592, 15.32 53.798)
```

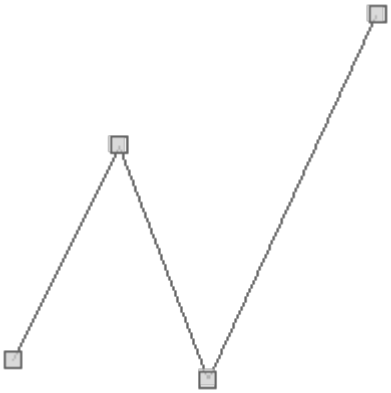
GeoJSON

Read a Geometry from GeoJSON using the GeoJSONReader

```
String json = '{"type":"Point","coordinates":[-123.15,46.237]}'
GeoJSONReader reader = new GeoJSONReader()
Geometry geometry = reader.read(json)
```

Read a Geometry from GeoJSON using the Geometry.fromGeoJSON() static method

```
String json =
'{"type":"LineString","coordinates":[[3.198,43.164],[6.713,49.755],[9.702,42.592],[15.
32,53.798]]}'
Geometry geometry = Geometry.fromGeoJSON(json)
```



Get the GeoJSON of a Geometry

```
Geometry geometry = new Point(-123.15, 46.237)
String json = geometry.geoJSON
println json
```

```
{"type":"Point","coordinates":[-123.15,46.237]}
```

Write a Geometry to GeoJSON using the GeoJSONWriter

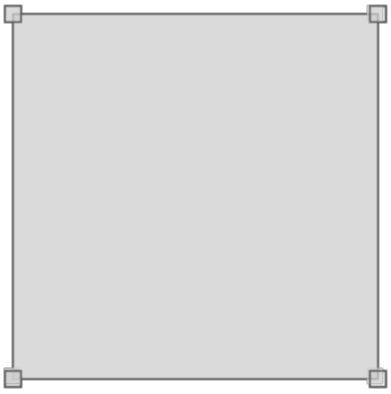
```
Geometry geometry = new LineString(
        [3.198, 43.164],
        [6.713, 49.755],
        [9.702, 42.592],
        [15.32, 53.798]
)
GeoJSONWriter writer = new GeoJSONWriter()
String json = writer.write(geometry)
println json
```

```
{"type":"LineString","coordinates":[[3.198,43.164],[6.713,49.755],[9.702,42.592],[15.3
2,53.798]]}
```

Creating Bounds

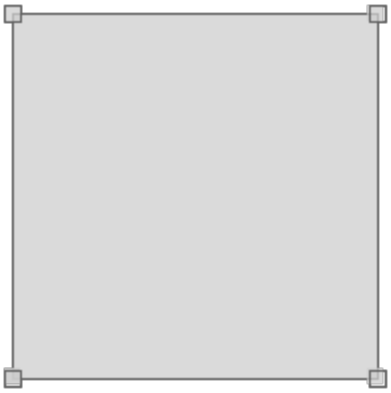
Create a Bounds from four coordinates (minx, miny, maxx, maxy) and a projection.

```
Bounds bounds = new Bounds(-127.265, 43.068, -113.554, 50.289, "EPSG:4326")
```



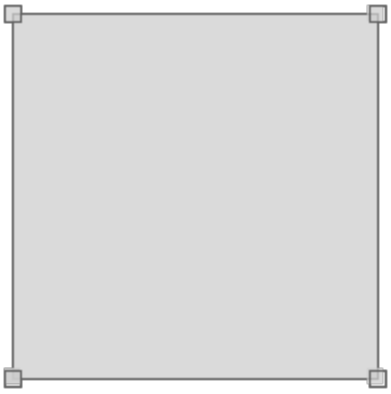
Create a Bounds from four coordinates (minx, miny, maxx, maxy) without a projection. The projection can be set later.

```
Bounds bounds = new Bounds(-127.265, 43.068, -113.554, 50.289)
bounds.proj = new Projection("EPSG:4326")
```



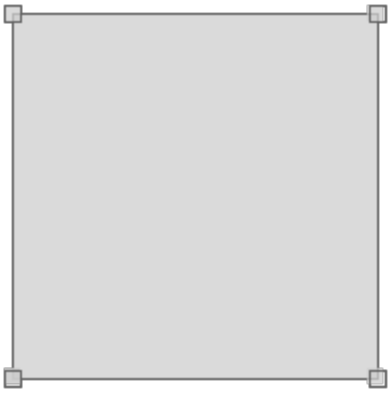
Create a Bounds from a string with commas delimiting minx, miny, maxx, maxy and projection values.

```
Bounds bounds = Bounds.fromString("-127.265,43.068,-113.554,50.289,EPSG:4326")
```



Create a Bounds from a string with spaces delimiting minx, miny, maxx, maxy and projection values.

```
Bounds bounds = Bounds.fromString("12.919921874999998 40.84706035607122 15.99609375
41.77131167976407 EPSG:4326")
```



Getting Bounds Properties

Create a Bounds and view it's string representation

```
Bounds bounds = new Bounds(-127.265, 43.068, -113.554, 50.289, "EPSG:4326")
String boundsStr = bounds.toString()
println boundsStr
```

```
(-127.265,43.068,-113.554,50.289,EPSG:4326)
```

Get the minimum x coordinate

```
double minX = bounds.minX
println minX
```

```
-127.265
```

Get the minimum y coordinate

```
double minY = bounds.minY
println minY
```

```
43.068
```

Get the maximum x coordinate

```
double maxX = bounds.maxX
println maxX
```

```
-113.554
```

Get the maximum y coordinate

```
double maxY = bounds.maxY
println maxY
```

```
50.289
```

Get the Projection

```
Projection proj = bounds.proj
println proj.id
```

```
EPSG:4326
```

Get the area

```
double area = bounds.area
println area
```

```
99.00713100000004
```

Get the width

```
double width = bounds.width
println width
```

```
13.710999999999999
```

Get the height

```
double height = bounds.height
println height
```

```
7.221000000000004
```

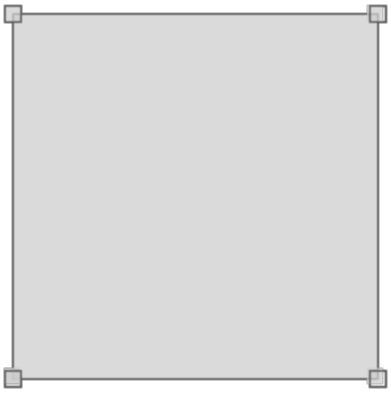
Get the aspect ratio

```
double aspect = bounds.aspect
println aspect
```

```
1.8987674837280144
```

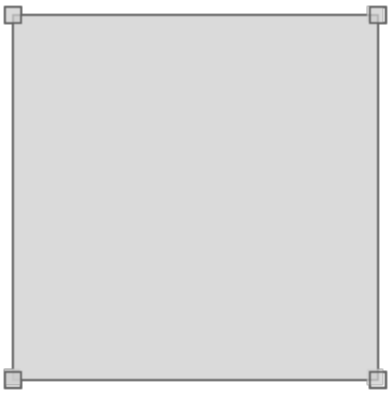
A Bounds is not a Geometry but you can get a Geometry from a Bounds

```
Bounds bounds = new Bounds(-122.485, 47.246, -122.452, 47.267, "EPSG:4326")
Geometry geometry = bounds.geometry
```



You can also get a Polygon from a Bounds

```
Bounds bounds = new Bounds(-122.485, 47.246, -122.452, 47.267, "EPSG:4326")
Polygon polygon = bounds.polygon
```



Get the four corners from a Bounds as a List of Points

```
Bounds bounds = new Bounds(-122.485, 47.246, -122.452, 47.267, "EPSG:4326")
List<Point> points = bounds.corners
```

Processing Bounds

Reproject a Bounds from one Projection to another.

```
Bounds bounds = new Bounds(-122.485, 47.246, -122.452, 47.267, "EPSG:4326")
println bounds
```

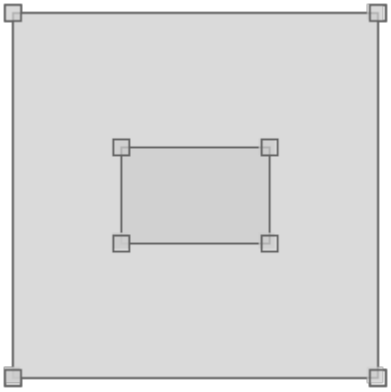
```
(-122.485,47.246,-122.452,47.267,EPSG:4326)
```

```
Bounds reprojectedBounds = bounds.reproject("EPSG:2927")
println reprojectedBounds
```

```
(1147444.7684517875,703506.223164177,1155828.120242509,711367.9403610165,EPSG:2927)
```

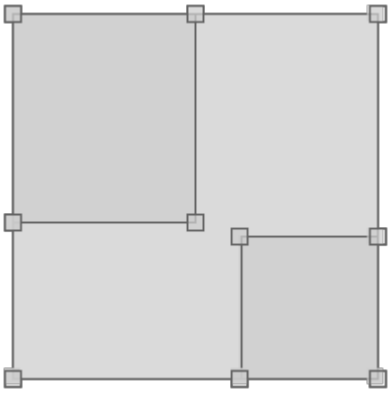
Expand a Bounds by a given distance

```
Bounds bounds1 = new Bounds(-127.265, 43.068, -113.554, 50.289, "EPSG:4326")
Bounds bounds2 = new Bounds(-127.265, 43.068, -113.554, 50.289, "EPSG:4326")
bounds2.expandBy(10.1)
```



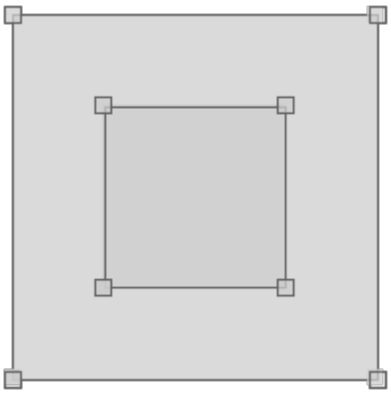
Expand a Bounds to include another Bounds

```
Bounds bounds1 = new Bounds(8.4375, 37.996162679728116, 19.6875, 46.07323062540835,
"EPSG:4326")
Bounds bounds2 = new Bounds(22.5, 31.952162238024975, 30.937499999999996,
37.43997405227057, "EPSG:4326")
bounds1.expand(bounds2)
```



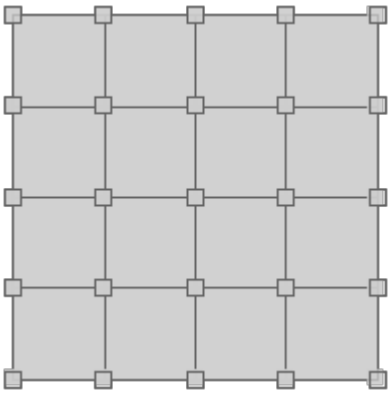
Scale an existing Bounds some distance to create a new Bounds

```
Bounds bounds1 = new Bounds(-127.265, 43.068, -113.554, 50.289, "EPSG:4326")
Bounds bounds2 = bounds1.scale(2)
```



Divide a Bounds into smaller tiles or Bounds

```
Bounds bounds = new Bounds(-122.485, 47.246, -122.452, 47.267, "EPSG:4326")
List<Bounds> subBounds = bounds.tile(0.25)
```



Calculate a quad tree for this Bounds between the start and stop levels. A Closure is called for each new Bounds generated.

```
Bounds bounds = new Bounds(-180, -90, 180, 90, "EPSG:4326")
bounds.quadTree(0,2) { Bounds b ->
    println b
}
```

```
(-180.0,-90.0,180.0,90.0,EPSG:4326)
(-180.0,-90.0,0.0,0.0,EPSG:4326)
(-180.0,0.0,0.0,90.0,EPSG:4326)
(0.0,-90.0,180.0,0.0,EPSG:4326)
(0.0,0.0,180.0,90.0,EPSG:4326)
```

Determine whether a Bounds is empty or not. A Bounds is empty if it is null or it's area is 0.

```
Bounds bounds = new Bounds(0,10,10,20)
println bounds.isEmpty()
```

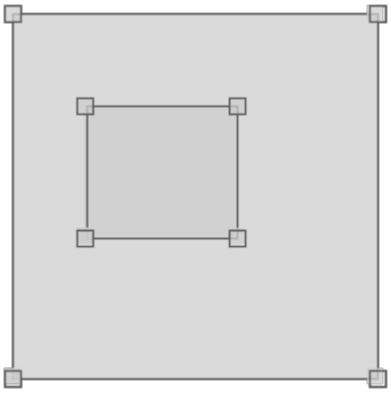
```
false
```

```
Bounds emptyBounds = new Bounds(0,10,10,10)
println emptyBounds.isEmpty()
```

```
true
```

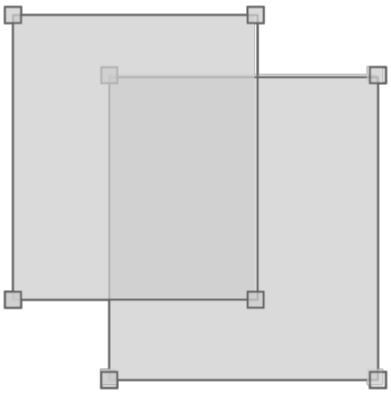
Determine if a Bounds contains another Bounds

```
Bounds bounds1 = new Bounds(-107.226, 34.597, -92.812, 43.068)
Bounds bounds2 = new Bounds(-104.326, 37.857, -98.349, 40.913)
println bounds1.contains(bounds2)
```



```
true
```

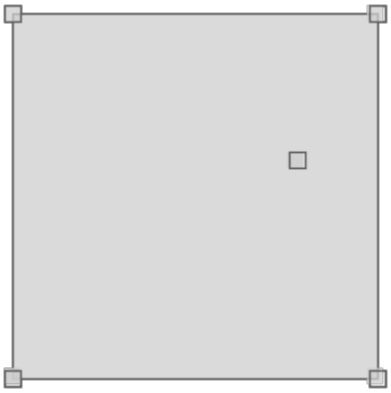
```
Bounds bounds3 = new Bounds(-112.412, 36.809, -99.316, 44.777)
println bounds1.contains(bounds3)
```



```
false
```

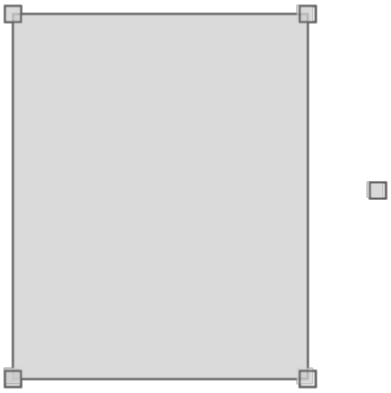
Determine if a Bounds contains a Point

```
Bounds bounds = new Bounds(-107.226, 34.597, -92.812, 43.068)
Point point1 = new Point(-95.976, 39.639)
println bounds.contains(point1)
```



```
true
```

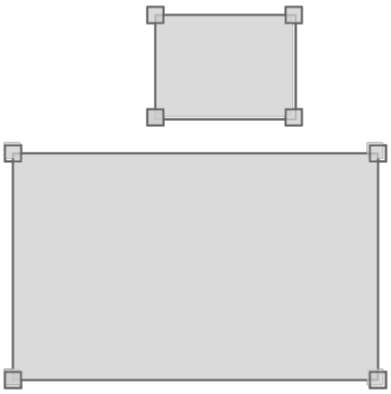
```
Point point2 = new Point(-89.384, 38.959)
println bounds.contains(point2)
```



```
true
```

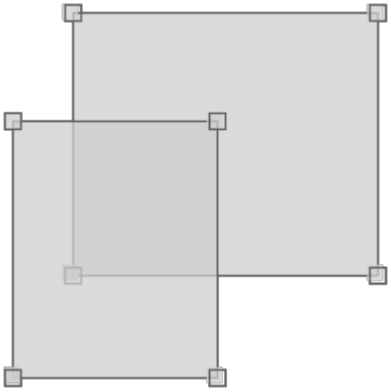
Determine if two Bounds intersect

```
Bounds bounds1 = new Bounds(-95.885, 46.765, -95.788, 46.811)
Bounds bounds2 = new Bounds(-95.847, 46.818, -95.810, 46.839)
println bounds1.intersects(bounds2)
```



```
false
```

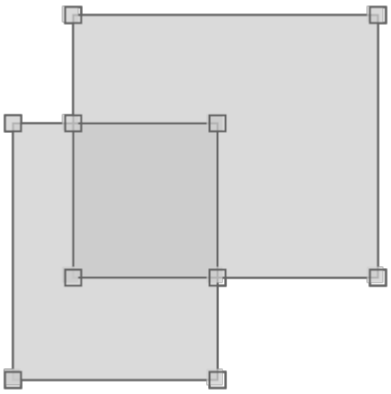
```
Bounds bounds3 = new Bounds(-95.904, 46.747, -95.839, 46.792)
println bounds1.intersects(bounds3)
```



```
true
```

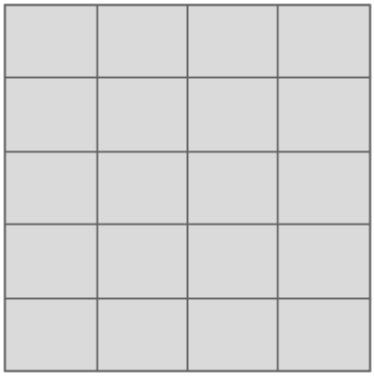
Calculate the intersection between two Bounds

```
Bounds bounds1 = new Bounds(-95.885, 46.765, -95.788, 46.811)
Bounds bounds2 = new Bounds(-95.904, 46.747, -95.839, 46.792)
Bounds bounds3 = bounds1.intersection(bounds2)
```



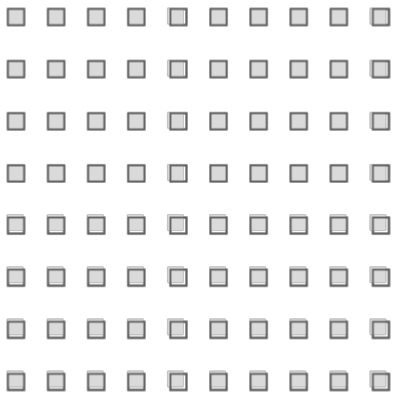
Generate a grid from a Bounds with a given number of columns and rows and the polygon shape. Other shapes include: polygon, point, circle/ellipse, hexagon, hexagon-inv).

```
Bounds bounds = new Bounds(-180,-90,180,90,"EPSG:4326")
Geometry geometry = bounds.getGrid(5,4,"polygon")
```



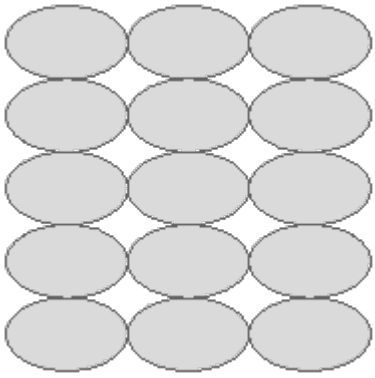
Generate a grid from a Bounds with a given number of columns and rows and a point shape. A Closure that is called with a geometry, column, and row for each grid cell that is created.

```
Bounds bounds = new Bounds(-180,-90,180,90,"EPSG:4326")
List geometries = []
Geometry geometry = bounds.generateGrid(10,8,"point") { Geometry g, int col, int row
->
    geometries.add(g)
}
```



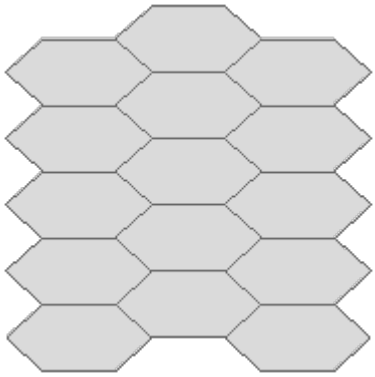
Generate a grid from a Bounds with a given cell width and height and a circle/ellipse shape.

```
Bounds bounds = new Bounds(-180,-90,180,90,"EPSG:4326")
Geometry geometry = bounds.getGrid(72.0,72.0,"circle")
```



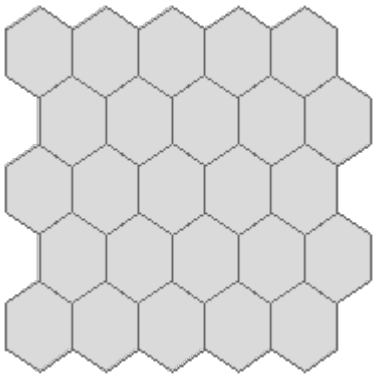
Generate a grid from a Bounds with a given cell width and height and a hexagon shape. A Closure is called with a geometry, column, and row for each grid cell generated.

```
Bounds bounds = new Bounds(-180,-90,180,90,"EPSG:4326")
List geometries = []
Geometry geometry = bounds.generateGrid(72.0,72.0,"hexagon") { Geometry g, int col,
int row ->
    geometries.add(g)
}
```



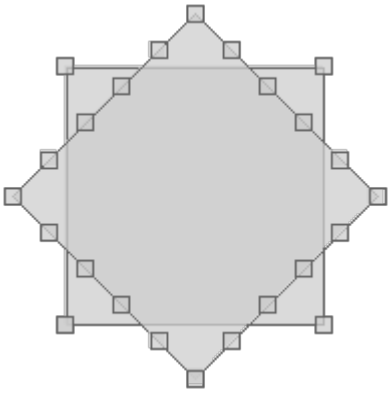
Generate a grid from a Bounds with a given cell width and height and an inverted hexagon shape.

```
Bounds bounds = new Bounds(-180,-90,180,90,"EPSG:4326")
Geometry geometry = bounds.getGrid(5,5,"hexagon-inv")
```



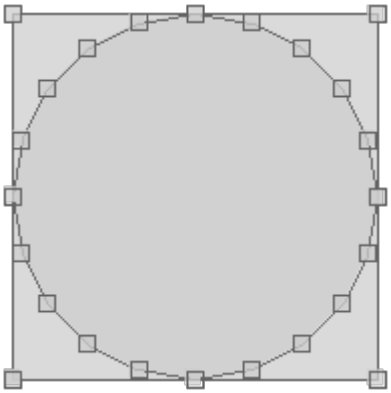
Create a rectangle from a Bounds with a given number of Points and a rotation angle in radians.

```
Bounds bounds = new Bounds(0,0,20,20)
Polygon polygon = bounds.createRectangle(20,Math.toRadians(45))
```



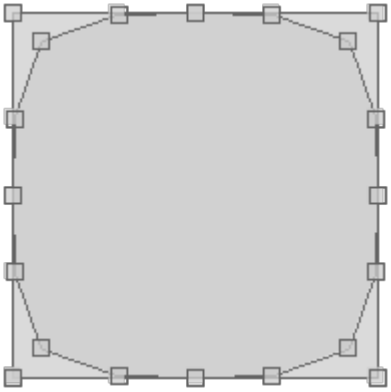
Create an ellipse from a Bounds. The default number of points is 20 and the default rotation angle in radians is 0.

```
Bounds bounds = new Bounds(0,0,20,20)
Polygon polygon = bounds.createEllipse()
```



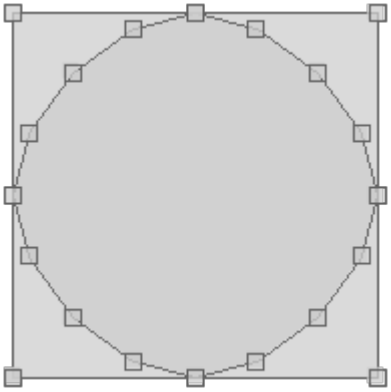
Create a squircle from a Bounds. The default number of points is 20 and the default rotation angle in radians is 0.

```
Bounds bounds = new Bounds(0,0,20,20)
Polygon polygon = bounds.createSquircle()
```



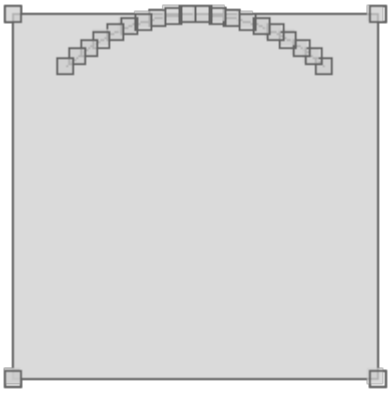
Create a super circle from a Bounds with a given power. The default number of points is 20 and the default rotation angle in radians is 0.

```
Bounds bounds = new Bounds(0,0,20,20)
Polygon polygon = bounds.createSuperCircle(1.75)
```



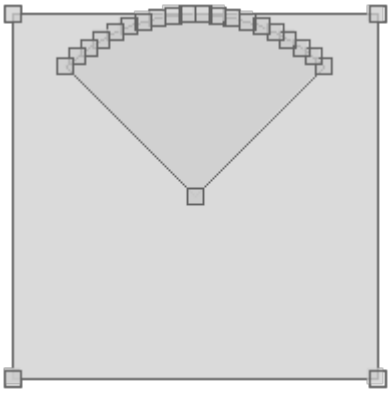
Create an arc from a Bounds with a start angle and angle extent. The default number of points is 20 and the default rotation angle in radians is 0.

```
Bounds bounds = new Bounds(0,0,20,20)
LineString lineString = bounds.createArc(Math.toRadians(45), Math.toRadians(90))
```



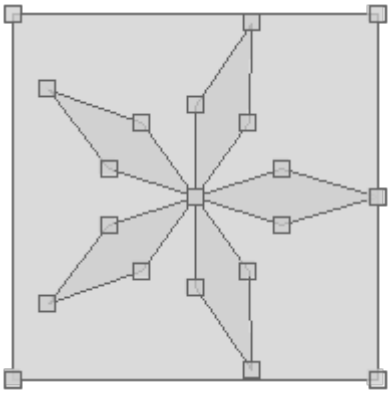
Create an arc polygon from a Bounds with a start angle and angle extent. The default number of points is 20 and the default rotation angle in radians is 0.

```
Bounds bounds = new Bounds(0,0,20,20)
Polygon polygon = bounds.createArcPolygon(Math.toRadians(45), Math.toRadians(90))
```



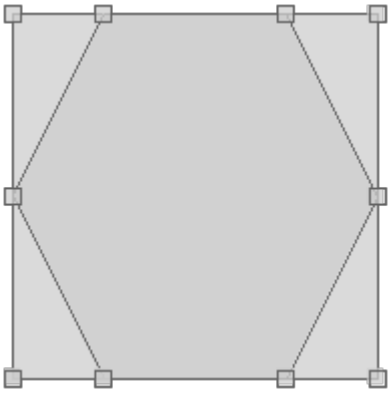
Create a sine star from a Bounds with a number of arms and an arm length ratio. The default number of points is 20 and the default rotation angle in radians is 0.

```
Bounds bounds = new Bounds(0,0,20,20)
Polygon polygon = bounds.createSineStar(5, 2.3)
```



Create a hexagon from a Bounds that is either inverted (false) or not (true).

```
Bounds bounds = new Bounds(0,0,20,20)
Polygon polygon = bounds.createHexagon(false)
```



Projection Recipes

Creating Projections

Create a Projection from an EPSG Code

```
Projection proj = new Projection("EPSG:4326")
println proj.wkt
```

```
GEOGCS["WGS 84",
  DATUM["World Geodetic System 1984",
    SPHEROID["WGS 84", 6378137.0, 298.257223563, AUTHORITY["EPSG","7030"]],
    AUTHORITY["EPSG","6326"]],
  PRIMEM["Greenwich", 0.0, AUTHORITY["EPSG","8901"]],
  UNIT["degree", 0.017453292519943295],
  AXIS["Geodetic longitude", EAST],
  AXIS["Geodetic latitude", NORTH],
  AUTHORITY["EPSG","4326"]]
```

Create a Projection from a WKT Projection String

```
      Projection proj = new Projection("""GEOGCS["WGS 84",
DATUM["World Geodetic System 1984",
  SPHEROID["WGS 84", 6378137.0, 298.257223563, AUTHORITY["EPSG","7030"]],
  AUTHORITY["EPSG","6326"]],
PRIMEM["Greenwich", 0.0, AUTHORITY["EPSG","8901"]],
UNIT["degree", 0.017453292519943295],
AXIS["Geodetic longitude", EAST],
AXIS["Geodetic latitude", NORTH],
AUTHORITY["EPSG","4326"]]""")
```

```
GEOGCS["WGS 84",
  DATUM["World Geodetic System 1984",
    SPHEROID["WGS 84", 6378137.0, 298.257223563, AUTHORITY["EPSG","7030"]],
    AUTHORITY["EPSG","6326"]],
  PRIMEM["Greenwich", 0.0, AUTHORITY["EPSG","8901"]],
  UNIT["degree", 0.017453292519943295],
  AXIS["Geodetic longitude", EAST],
  AXIS["Geodetic latitude", NORTH],
  AUTHORITY["EPSG","4326"]]
```

Create a Projection from well known name

```
Projection proj = new Projection("Mollweide")
println proj.wkt
```

```
PROJCS["Mollweide",
  GEOGCS["WGS84",
    DATUM["WGS84",
      SPHEROID["WGS84", 6378137.0, 298.257223563]],
    PRIMEM["Greenwich", 0.0],
    UNIT["degree", 0.017453292519943295],
    AXIS["Longitude", EAST],
    AXIS["Latitude", NORTH]],
  PROJECTION["Mollweide"],
  PARAMETER["semi-minor axis", 6378137.0],
  PARAMETER["Longitude of natural origin", 0.0],
  UNIT["m", 1.0],
  AXIS["Easting", EAST],
  AXIS["Northing", NORTH]]
```

Get a List of all supported Projections (this is really slow)

```
List<Projection> projections = Projection.projections()
```

```
EPSG:4326
EPSG:4269
EPSG:26918
EPSG:2263
EPSG:2927
...
```

Getting Projection Properties

Get the id

```
Projection proj = new Projection("EPSG:4326")
String id = proj.id
```

```
EPSG:4326
```

Get the srs

```
String srs = proj.srs
```

```
EPSG:4326
```

Get the epsg code

```
int epsg = proj.epsg
```

```
4326
```

Get the WKT

```
String wkt = proj.wkt
```

```
GEOGCS["WGS 84",
  DATUM["World Geodetic System 1984",
    SPHEROID["WGS 84", 6378137.0, 298.257223563, AUTHORITY["EPSG","7030"]],
    AUTHORITY["EPSG","6326"]],
  PRIMEM["Greenwich", 0.0, AUTHORITY["EPSG","8901"]],
  UNIT["degree", 0.017453292519943295],
  AXIS["Geodetic longitude", EAST],
  AXIS["Geodetic latitude", NORTH],
  AUTHORITY["EPSG","4326"]]
```

Get the Bounds in the native Projection

```
Bounds bounds = proj.bounds
```

```
(-180.0,-90.0,180.0,90.0,EPSG:4326)
```

Get the Bounds in the EPSG:4326

```
Bounds geoBounds = proj.geoBounds
```

```
(-180.0,-90.0,180.0,90.0,EPSG:4326)
```

Using Projections

Transform a Geometry from one projection to another using the Projection static method with strings

```
Geometry epsg4326Geom = new Point(-122.440, 47.245)
Geometry epsg2927Geom = Projection.transform(epsg4326Geom, "EPSG:4326", "EPSG:2927")
println epsg2927Geom
```

```
POINT (1158609.2040371667 703068.0661327887)
```

Transform a Geometry from one projection to another using the Projection static method with Projections

```
Projection epsg4326 = new Projection("EPSG:4326")
Projection epsg2927 = new Projection("EPSG:2927")
Geometry epsg4326Geom = new Point(-122.440, 47.245)
Geometry epsg2927Geom = Projection.transform(epsg4326Geom, epsg4326, epsg2927)
println epsg2927Geom
```

```
POINT (1158609.2040371667 703068.0661327887)
```

Transform a Geometry from one projection to another using two Projections

```
Projection fromProj = new Projection("EPSG:4326")
Projection toProj = new Projection("EPSG:2927")
Geometry geom = new Point(-122.440, 47.245)
Geometry projectedGeom = fromProj.transform(geom, toProj)
println projectedGeom
```

```
POINT (1158609.2040371667 703068.0661327887)
```

Transform a Geometry from one projection to another using a Projections and a String

```
Projection fromProj = new Projection("EPSG:4326")
Geometry geom = new Point(-122.440, 47.245)
Geometry projectedGeom = fromProj.transform(geom, "EPSG:2927")
println projectedGeom
```

```
POINT (1158609.2040371667 703068.0661327887)
```

Using Geodetic

Create a Geodetic object with an ellipsoid

```
Geodetic geodetic = new Geodetic("wgs84")
println geodetic
```

```
Geodetic [SPHEROID["WGS 84", 6378137.0, 298.257223563]]
```

Calculate the forward and back azimuth and distance between the given two Points.

```
Geodetic geodetic = new Geodetic("clrk66")
Point bostonPoint = new Point(-71.117, 42.25)
Point portlandPoint = new Point(-123.683, 45.52)
Map results = geodetic.inverse(bostonPoint, portlandPoint)
double forwardAzimuth =  results.forwardAzimuth
println forwardAzimuth
```

```
-66.52547810974724
```

```
double backAzimuth = results.backAzimuth
println backAzimuth
```

```
75.65817457195088
```

```
double distance = results.distance
println distance
```

```
4164050.4598800642
```

Calculate a new Point and back azimuth given the starting Point, azimuth, and distance.

```
Geodetic geodetic = new Geodetic("clrk66")
Point bostonPoint = new Point(-71.117, 42.25)
Map results = geodetic.forward(bostonPoint, -66.531, 4164192.708)
Point point = results.point
println point
```

```
POINT (-123.6835797667373 45.516427795897236)
```

```
double azimuth = results.backAzimuth
println azimuth
```

```
75.65337425050724
```

Place the given number of points between starting and ending Points

```
Geodetic geodetic = new Geodetic("clrk66")
Point bostonPoint = new Point(-71.117, 42.25)
Point portlandPoint = new Point(-123.683, 45.52)
List<Point> points = geodetic.placePoints(bostonPoint, portlandPoint, 10)
points.each { Point point ->
    println point.wkt
}
```

```
POINT (-75.41357382496236 43.52791689304304)
POINT (-79.8828640042499 44.63747566950249)
POINT (-84.51118758826816 45.565540142641005)
POINT (-89.27793446221685 46.300124344169255)
POINT (-94.15564606698499 46.83102721803566)
POINT (-99.11079892605703 47.15045006457598)
POINT (-104.10532353179985 47.25351783423774)
POINT (-109.09873812691617 47.13862709798196)
POINT (-114.05062990603696 46.80756425557422)
POINT (-118.92312608779855 46.26537395700513)
```

Using Decimal Degrees

Create a new DecimalDegrees from a longitude and latitude

```
DecimalDegrees decimalDegrees = new DecimalDegrees(-122.525619, 47.212023)
println decimalDegrees
```

```
-122° 31' 32.2284" W, 47° 12' 43.2828" N
```

Create a new DecimalDegrees from a Point

```
DecimalDegrees decimalDegrees = new DecimalDegrees(new Point(-122.525619,47.212023))
println decimalDegrees
```

```
POINT (-122.52561944444444 47.212022222222224)
```

Create a new DecimalDegrees from a Longitude and Latitude string

```
DecimalDegrees decimalDegrees = new DecimalDegrees("-122.525619, 47.212023")
println decimalDegrees
```

```
-122° 31' 32.2284" W, 47° 12' 43.2828" N
```

Create a new DecimalDegrees from two strings with glyphs

```
DecimalDegrees decimalDegrees = new DecimalDegrees("122\u00B0 31' 32.23\" W",
"47\u00B0 12' 43.28\" N")
println decimalDegrees
```

```
-122° 31' 32.2300" W, 47° 12' 43.2800" N
```

Create a new DecimalDegrees from two strings

```
DecimalDegrees decimalDegrees = new DecimalDegrees("122d 31m 32.23s W", "47d 12m
43.28s N")
println decimalDegrees
```

```
-122° 31' 32.2300" W, 47° 12' 43.2800" N
```

Create a new DecimalDegrees from a single Degrees Minutes Seconds formatted string

```
DecimalDegrees decimalDegrees = new DecimalDegrees("122d 31m 32.23s W, 47d 12m 43.28s
N")
println decimalDegrees
```

```
-122° 31' 32.2300" W, 47° 12' 43.2800" N
```

Create a new DecimalDegrees from a single Decimal Degree Minutes formatted string with glyphs

```
DecimalDegrees decimalDegrees = new DecimalDegrees("122\u00B0 31.5372' W, 47\u00B0
12.7213' N")
println decimalDegrees
```

```
-122° 31' 32.2320" W, 47° 12' 43.2780" N
```

Create a new DecimalDegrees from a single Decimal Degree Minutes formatted string

```
DecimalDegrees decimalDegrees = new DecimalDegrees("122d 31.5372m W, 47d 12.7213m N")
println decimalDegrees
```

```
-122° 31' 32.2320" W, 47° 12' 43.2780" N
```

Get degrees minutes seconds from a DecimalDegrees object

```
DecimalDegrees decimalDegrees = new DecimalDegrees("122d 31m 32.23s W", "47d 12m
43.28s N")
Map dms = decimalDegrees.dms
println "Degrees: ${dms.longitude.degrees}"
println "Minutes: ${dms.longitude.minutes}"
println "Seconds: ${dms.longitude.seconds}"
```

```
Degrees: -122
Minutes: 31
Seconds: 32.22999999998388
```

```
println "Degrees: ${dms.latitude.degrees}"
println "Minutes: ${dms.latitude.minutes}"
println "Seconds: ${dms.latitude.seconds}"
```

```
Degrees: 47
Minutes: 12
Seconds: 43.280000000006396
```

Convert a DecimalDegrees object to a DMS String with glyphs

```
DecimalDegrees decimalDegrees = new DecimalDegrees("122d 31m 32.23s W", "47d 12m
43.28s N")
println decimalDegrees.toDms(true)
```

```
-122° 31' 32.2300" W, 47° 12' 43.2800" N
```

Convert a DecimalDegrees object to a DMS String without glyphs

```
println decimalDegrees.toDms(false)
```

```
-122d 31m 32.2300s W, 47d 12m 43.2800s N
```

Get degrees minutes from a DecimalDegrees object

```
DecimalDegrees decimalDegrees = new DecimalDegrees("122d 31m 32.23s W", "47d 12m
43.28s N")
Map dms = decimalDegrees.ddm
println "Degrees: ${dms.longitude.degrees}"
println "Minutes: ${dms.longitude.minutes}"
```

```
Degrees: -122
Minutes: 31.537166666666398
```

```
println "Degrees: ${dms.latitude.degrees}"
println "Minutes: ${dms.latitude.minutes}"
```

```
Degrees: 47
Minutes: 12.72133333333344
```

Convert a DecimalDegrees object to a DDM String with glyphs

```
DecimalDegrees decimalDegrees = new DecimalDegrees("122d 31m 32.23s W", "47d 12m
43.28s N")
println decimalDegrees.toDdm(true)
```

```
-122° 31.5372' W, 47° 12.7213' N
```

Convert a DecimalDegrees object to a DDM String without glyphs

```
println decimalDegrees.toDdm(false)
```

```
-122d 31.5372m W, 47d 12.7213m N
```

Get a Point from a DecimalDegrees object

```
DecimalDegrees decimalDegrees = new DecimalDegrees("122d 31m 32.23s W", "47d 12m
43.28s N")
Point point = decimalDegrees.point
```

```
POINT (-122.52561944444444 47.212022222222224)
```

Spatial Index Recipes

Using STRtree

Create a STRtree spatial index

```
STRtree index = new STRtree()
```

Insert Geometries and their Bounds

```
index.insert(new Bounds(0,0,10,10), new Point(5,5))
index.insert(new Bounds(2,2,6,6), new Point(4,4))
index.insert(new Bounds(20,20,60,60), new Point(30,30))
index.insert(new Bounds(22,22,44,44), new Point(32,32))
```

Get the size of the index

```
int size = index.size
println size
```

```
4
```

Query the index

```
List results = index.query(new Bounds(1,1,5,5))
results.each { Geometry geometry ->
    println geometry
}
```

```
POINT (4 4)
POINT (5 5)
```

Using Quadtree

Create a Quadtree spatial index

```
Quadtree index = new Quadtree()
```

Insert Geometries and their Bounds

```
index.insert(new Bounds(0,0,10,10), new Point(5,5))
index.insert(new Bounds(2,2,6,6), new Point(4,4))
index.insert(new Bounds(20,20,60,60), new Point(30,30))
index.insert(new Bounds(22,22,44,44), new Point(32,32))
```

Get the size of the index

```
int size = index.size
println size
```

```
4
```

Query the index with a Bounds

```
List results = index.query(new Bounds(1,1,5,5))
results.each { Geometry geometry ->
    println geometry
}
```

```
POINT (30 30)
POINT (32 32)
POINT (5 5)
POINT (4 4)
```

Query the entire index

```
List allResults = index.queryAll()
allResults.each { Geometry geometry ->
    println geometry
}
```

```
POINT (30 30)
POINT (32 32)
POINT (5 5)
POINT (4 4)
```

Remove an item from the index

```
Geometry itemToRemove = allResults[0]
boolean removed = index.remove(itemToRemove.bounds, itemToRemove)
println "Removed? ${removed}"
println "Size = ${index.size}"
```

```
Removed = true
Size = 3
```

Using GeoHash

Encode a Point as a String

```
GeoHash geohash = new GeoHash()
Point point = new Point(112.5584, 37.8324)
String hash = geohash.encode(point)
println hash
```

```
ww8p1r4t8
```

Decode a Point from a String

```
GeoHash geohash = new GeoHash()
Point point = geohash.decode("ww8p1r4t8")
println point
```

```
POINT (112.55838632583618 37.83238649368286)
```

Encode a Point as a Long

```
GeoHash geohash = new GeoHash()
Point point = new Point(112.5584, 37.8324)
long hash = geohash.encodeLong(point)
println long
```

```
4064984913515641
```

Decode a Point from a Long

```
GeoHash geohash = new GeoHash()
Point point = geohash.decode(4064984913515641)
println point
```

```
POINT (112.55839973688126 37.83240124583244)
```

Decode a Bounds from a String

```
GeoHash geohash = new GeoHash()
Bounds bounds = geohash.decodeBounds("ww8p1r4t8")
println bounds
```

```
(112.55836486816406,37.83236503601074,112.5584077835083,37.83240795135498)
```

Decode a Bounds from a Long

```
GeoHash geohash = new GeoHash()
Bounds bounds = geohash.decodeBounds(4064984913515641)
println bounds
```

```
(112.55836486816406,37.83236503601074,112.5584077835083,37.83240795135498)
```

Find neighboring geohash strings

```
GeoHash geohash = new GeoHash()
String hash = "dqcjq"
String north     = geohash.neighbor(hash, GeoHash.Direction.NORTH)
String northwest = geohash.neighbor(hash, GeoHash.Direction.NORTHWEST)
String west      = geohash.neighbor(hash, GeoHash.Direction.WEST)
String southwest = geohash.neighbor(hash, GeoHash.Direction.SOUTHWEST)
String south     = geohash.neighbor(hash, GeoHash.Direction.SOUTH)
String southeast = geohash.neighbor(hash, GeoHash.Direction.SOUTHEAST)
String east      = geohash.neighbor(hash, GeoHash.Direction.EAST)
String northeast = geohash.neighbor(hash, GeoHash.Direction.NORTHEAST)
String str = """
             | ${northwest} ${north} ${northeast}
             | ${west} ${hash} ${east}
             | ${southwest} ${south} ${southeast}
             |""".stripMargin()
println str
```

```
 dqcjt dqcjw dqcjx
 dqcjm dqcjq dqcjr
 dqcjj dqcjn dqcjp
```

Find neighboring geohash longs

```
GeoHash geohash = new GeoHash()
long hash = 1702789509
long north     = geohash.neighbor(hash, GeoHash.Direction.NORTH)
long northwest = geohash.neighbor(hash, GeoHash.Direction.NORTHWEST)
long west      = geohash.neighbor(hash, GeoHash.Direction.WEST)
long southwest = geohash.neighbor(hash, GeoHash.Direction.SOUTHWEST)
long south     = geohash.neighbor(hash, GeoHash.Direction.SOUTH)
long southeast = geohash.neighbor(hash, GeoHash.Direction.SOUTHEAST)
long east      = geohash.neighbor(hash, GeoHash.Direction.EAST)
long northeast = geohash.neighbor(hash, GeoHash.Direction.NORTHEAST)
String str = """
             | ${northwest} ${north} ${northeast}
             | ${west} ${hash} ${east}
             | ${southwest} ${south} ${southeast}
             |""".stripMargin()
println str
```

```
 1702789434 1702789520 1702789522
 1702789423 1702789509 1702789511
 1702789422 1702789508 1702789510
```

Find all neighboring geohash strings

```
GeoHash geohash = new GeoHash()
String hash = "dqcjq"
Map neighbors = geohash.neighbors(hash)
String north     = neighbors[GeoHash.Direction.NORTH]
String northwest = neighbors[GeoHash.Direction.NORTHWEST]
String west      = neighbors[GeoHash.Direction.WEST]
String southwest = neighbors[GeoHash.Direction.SOUTHWEST]
String south     = neighbors[GeoHash.Direction.SOUTH]
String southeast = neighbors[GeoHash.Direction.SOUTHEAST]
String east      = neighbors[GeoHash.Direction.EAST]
String northeast = neighbors[GeoHash.Direction.NORTHEAST]
String str = """
             | ${northwest} ${north} ${northeast}
             | ${west} ${hash} ${east}
             | ${southwest} ${south} ${southeast}
             |""".stripMargin()
println str
```

```
 dqcjt dqcjw dqcjx
 dqcjm dqcjq dqcjr
 dqcjj dqcjn dqcjp
```

Find all neighboring geohash longs

```
GeoHash geohash = new GeoHash()
long hash = 1702789509
Map neighbors = geohash.neighbors(hash)
long north     = neighbors[GeoHash.Direction.NORTH]
long northwest = neighbors[GeoHash.Direction.NORTHWEST]
long west      = neighbors[GeoHash.Direction.WEST]
long southwest = neighbors[GeoHash.Direction.SOUTHWEST]
long south     = neighbors[GeoHash.Direction.SOUTH]
long southeast = neighbors[GeoHash.Direction.SOUTHEAST]
long east      = neighbors[GeoHash.Direction.EAST]
long northeast = neighbors[GeoHash.Direction.NORTHEAST]
String str = """
             | ${northwest} ${north} ${northeast}
             | ${west} ${hash} ${east}
             | ${southwest} ${south} ${southeast}
             |""".stripMargin()
println str
```

```
 1702789434 1702789520 1702789522
 1702789423 1702789509 1702789511
 1702789422 1702789508 1702789510
```

Find all geohashes as strings within a Bounds

```
GeoHash geohash = new GeoHash()
List<String> bboxes = geohash.bboxes(new Bounds(120, 30, 120.0001, 30.0001), 8)
bboxes.each { String hash ->
    println hash
}
```

```
wtm6dtm6
wtm6dtm7
```

Find all geohashes as longs within a Bounds

```
GeoHash geohash = new GeoHash()
List<Long> bboxes = geohash.bboxesLong(new Bounds(120, 30, 120.0001, 30.0001), 40)
bboxes.each { long hash ->
    println hash
}
```

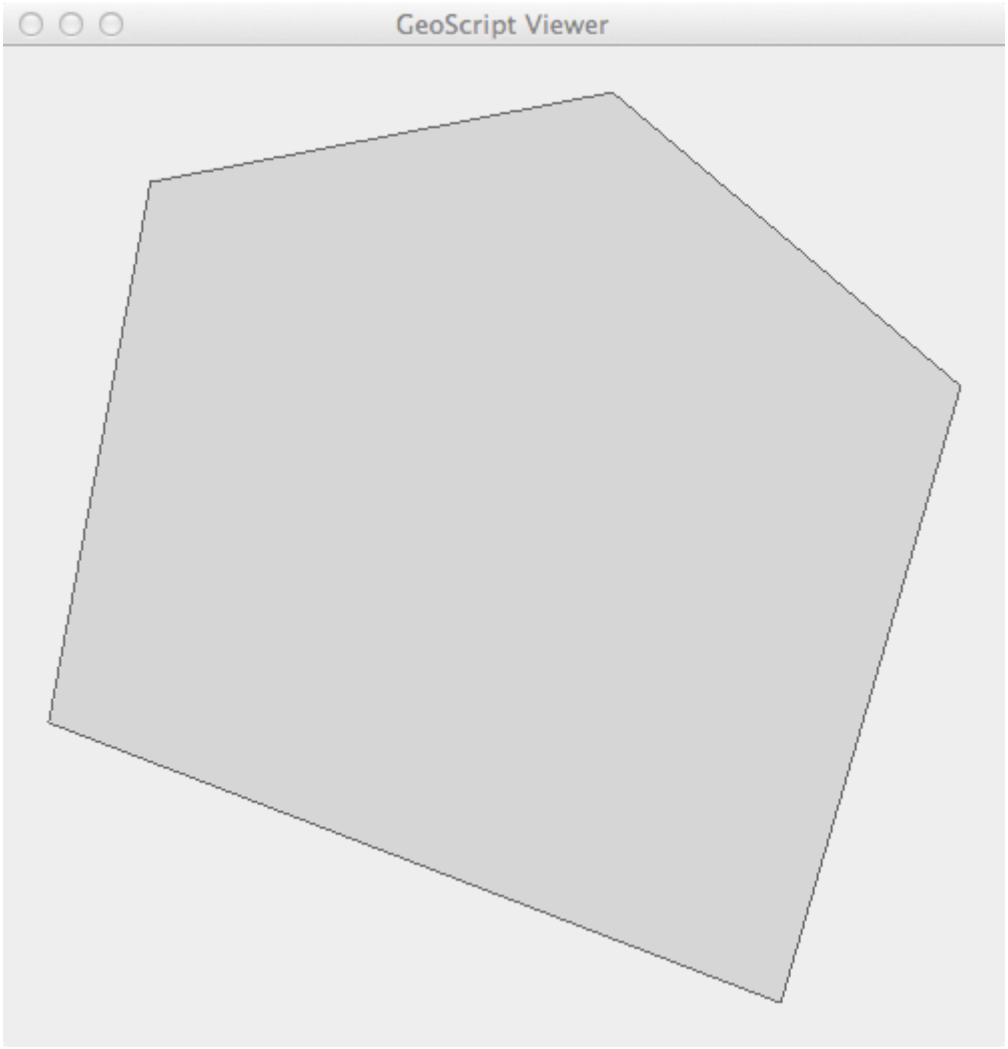
```
989560464998
989560464999
```

Viewer Recipes

Drawing geometries

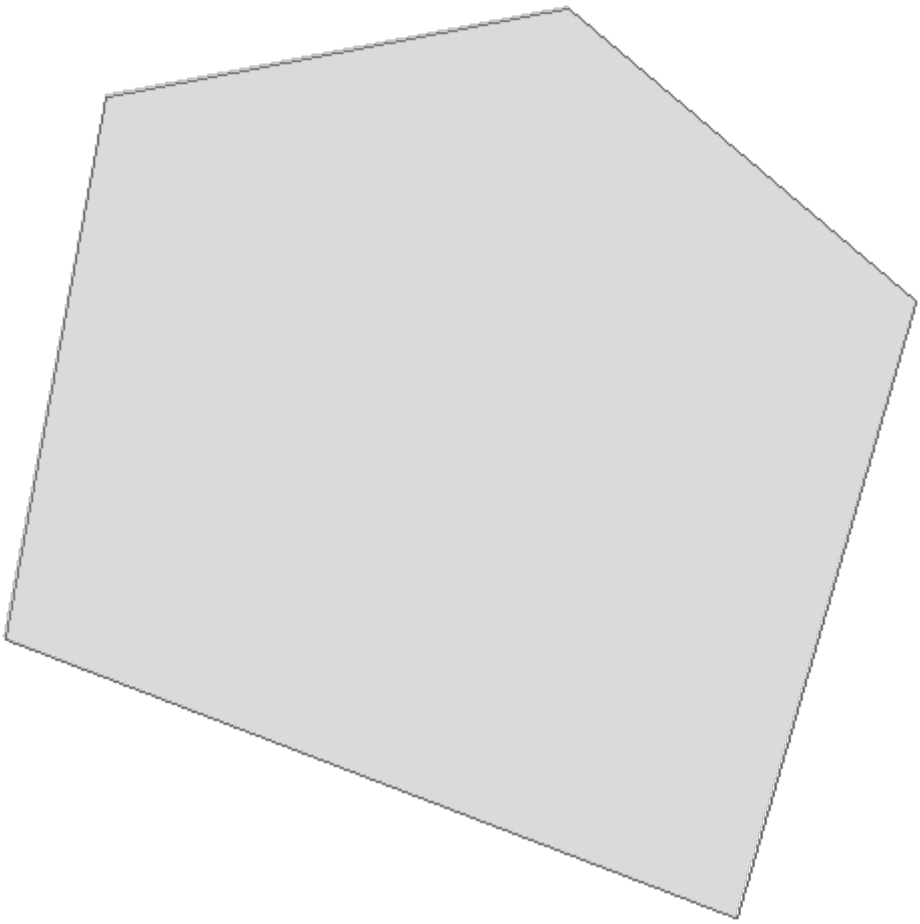
Draw a geometry in a simple GUI

```
Polygon polygon = new Polygon([[
       [-101.35986328125, 47.754097979680026],
       [-101.5576171875, 46.93526088057719],
       [-100.12939453125, 46.51351558059737],
       [-99.77783203125, 47.44294999517949],
       [-100.45898437499999, 47.88688085106901],
       [-101.35986328125, 47.754097979680026]
]])
Viewer.draw(polygon)
```



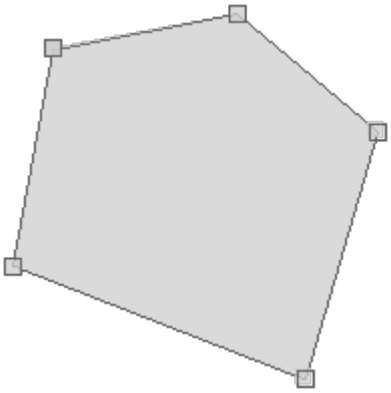
Draw a geometry to an image

```
Polygon polygon = new Polygon([[
       [-101.35986328125, 47.754097979680026],
       [-101.5576171875, 46.93526088057719],
       [-100.12939453125, 46.51351558059737],
       [-99.77783203125, 47.44294999517949],
       [-100.45898437499999, 47.88688085106901],
       [-101.35986328125, 47.754097979680026]
]])
BufferedImage image = Viewer.drawToImage(polygon)
```



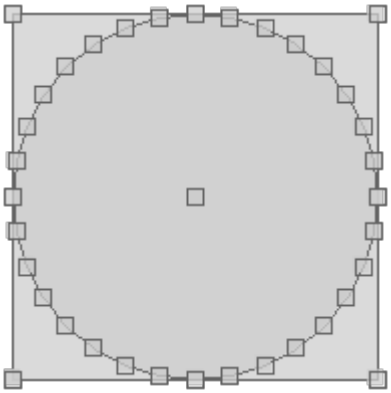
Draw a geometry to an image with options

```
Polygon polygon = new Polygon([[
       [-101.35986328125, 47.754097979680026],
       [-101.5576171875, 46.93526088057719],
       [-100.12939453125, 46.51351558059737],
       [-99.77783203125, 47.44294999517949],
       [-100.45898437499999, 47.88688085106901],
       [-101.35986328125, 47.754097979680026]
]])
BufferedImage image = Viewer.drawToImage(
        polygon,
        size: [200,200],
        drawCoords: true,
        fillCoords: true,
        fillPolys: true
)
```



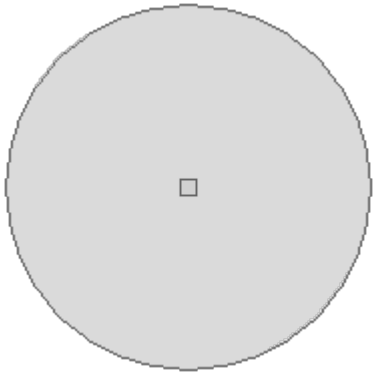
Draw a List of geometries to an image

```
Point point = new Point(-123.11, 47.23)
Geometry buffer = point.buffer(4)
Geometry bounds = buffer.bounds.geometry
BufferedImage image = Viewer.drawToImage(
        [bounds, buffer, point],
        size: [200,200],
        drawCoords: true,
        fillCoords: true,
        fillPolys: true
)
```



Draw a List of Geometries to a File

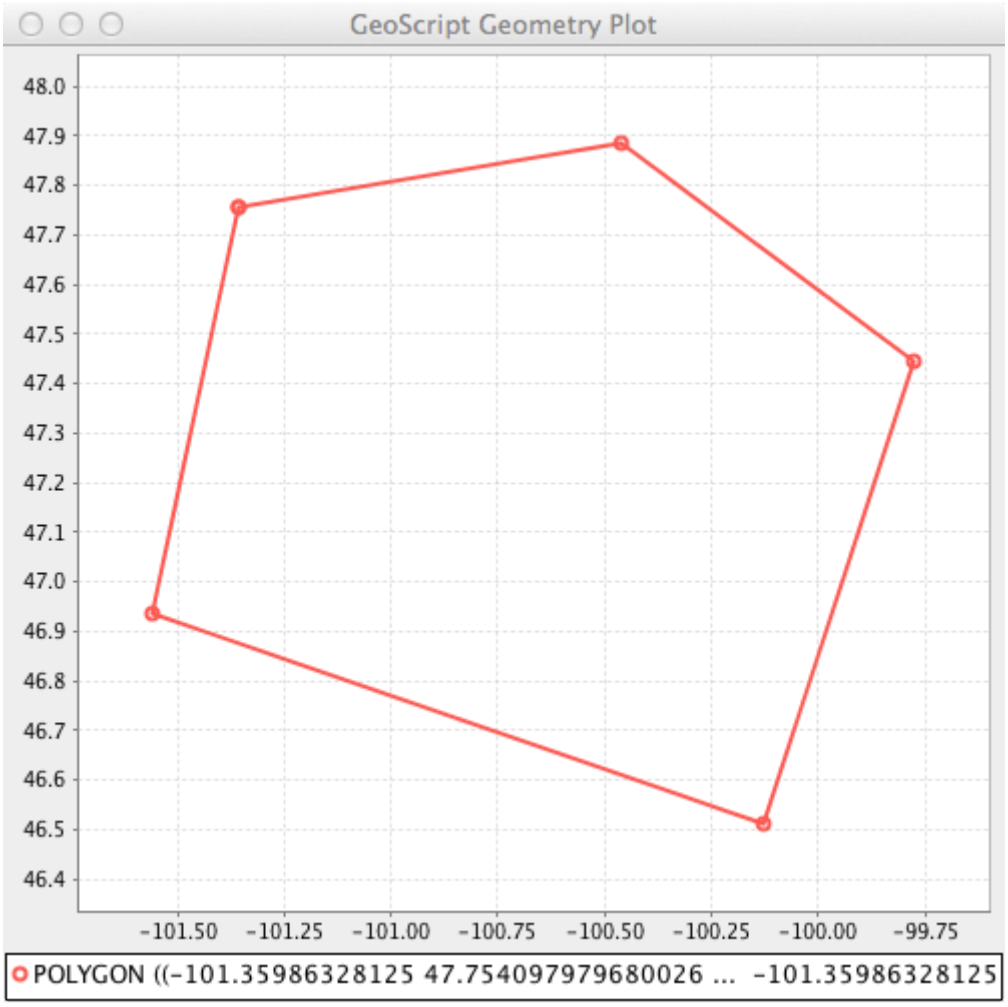
```
Point point = new Point(-123.11, 47.23)
Geometry buffer = point.buffer(4)
File file = new File("geometry.png")
Viewer.drawToFile([buffer, point], file, size: [200,200])
```



Plotting geometries

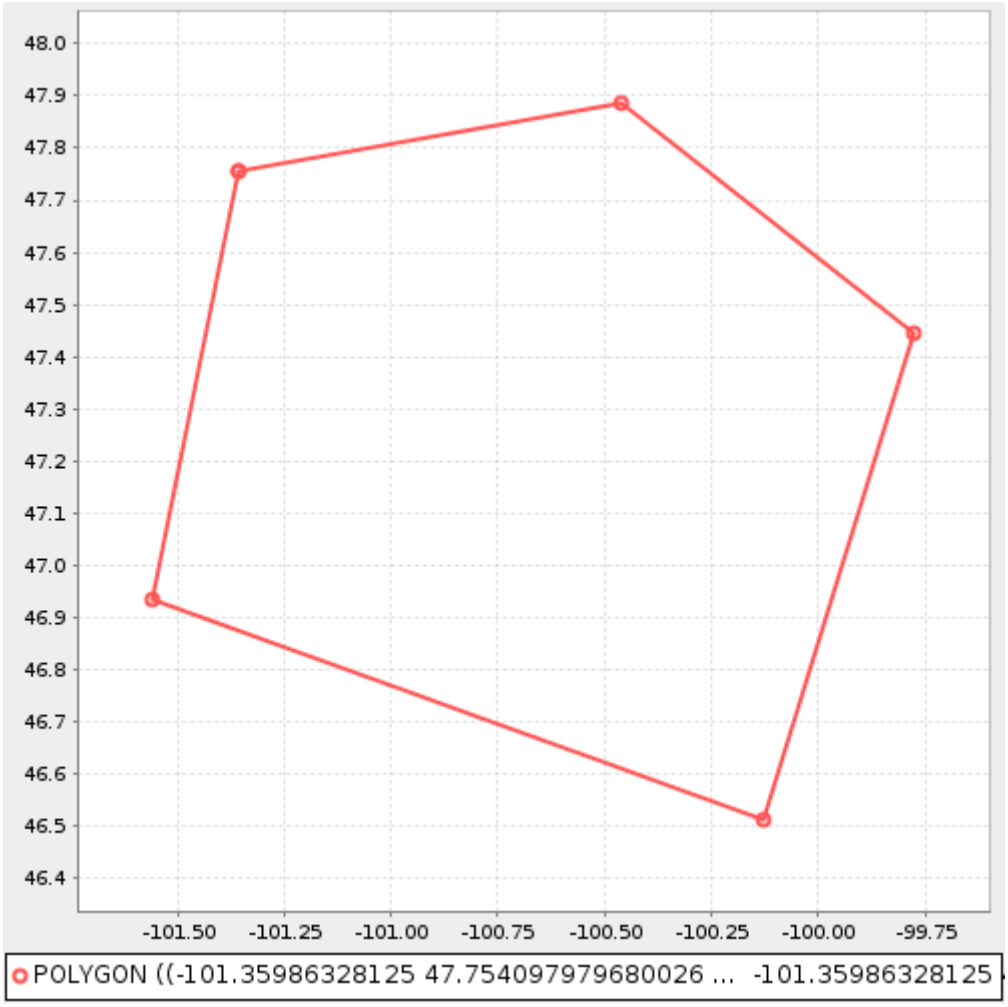
Plot a geometry in a simple GUI

```
Polygon polygon = new Polygon([[
       [-101.35986328125, 47.754097979680026],
       [-101.5576171875, 46.93526088057719],
       [-100.12939453125, 46.51351558059737],
       [-99.77783203125, 47.44294999517949],
       [-100.45898437499999, 47.88688085106901],
       [-101.35986328125, 47.754097979680026]
]])
Viewer.plot(polygon)
```



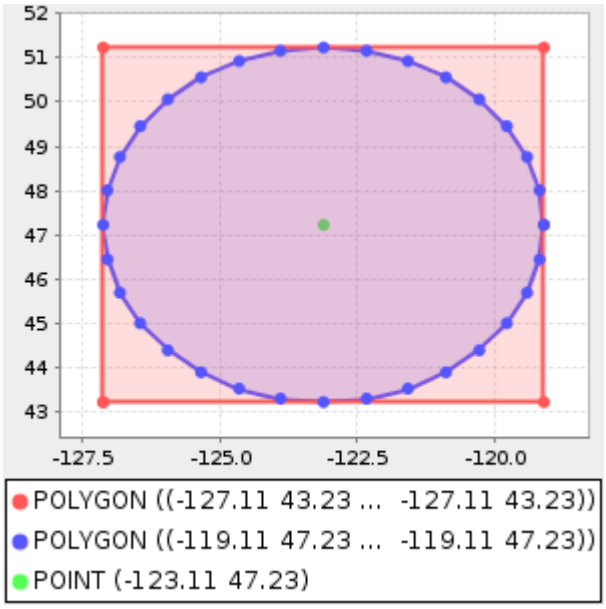
Plot a Geometry to an image

```
Polygon polygon = new Polygon([[
       [-101.35986328125, 47.754097979680026],
       [-101.5576171875, 46.93526088057719],
       [-100.12939453125, 46.51351558059737],
       [-99.77783203125, 47.44294999517949],
       [-100.45898437499999, 47.88688085106901],
       [-101.35986328125, 47.754097979680026]
]])
BufferedImage image = Viewer.plotToImage(polygon)
```



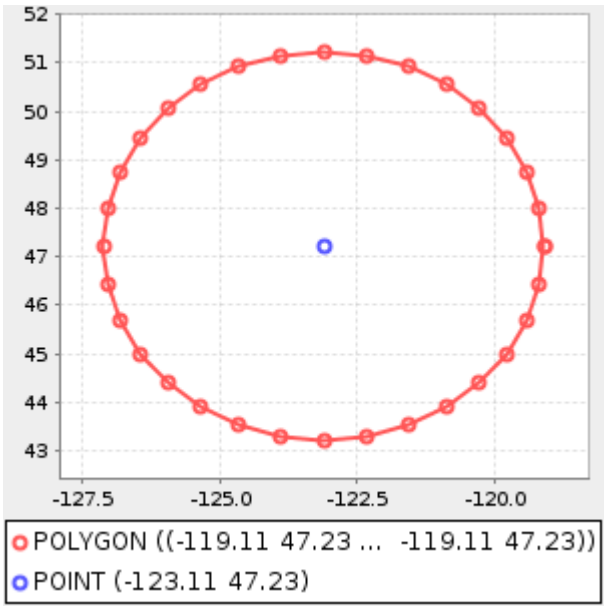
Plot a List of Geometries to an image

```
Point point = new Point(-123.11, 47.23)
Geometry buffer = point.buffer(4)
Geometry bounds = buffer.bounds.geometry
BufferedImage image = Viewer.plotToImage(
        [bounds, buffer, point],
        size: [300,300],
        drawCoords: true,
        fillCoords: true,
        fillPolys: true
)
```



Plot a Geometry to a File

```
Point point = new Point(-123.11, 47.23)
Geometry buffer = point.buffer(4)
File file = new File("geometry.png")
Viewer.plotToFile([buffer, point], file, size: [300,300])
```

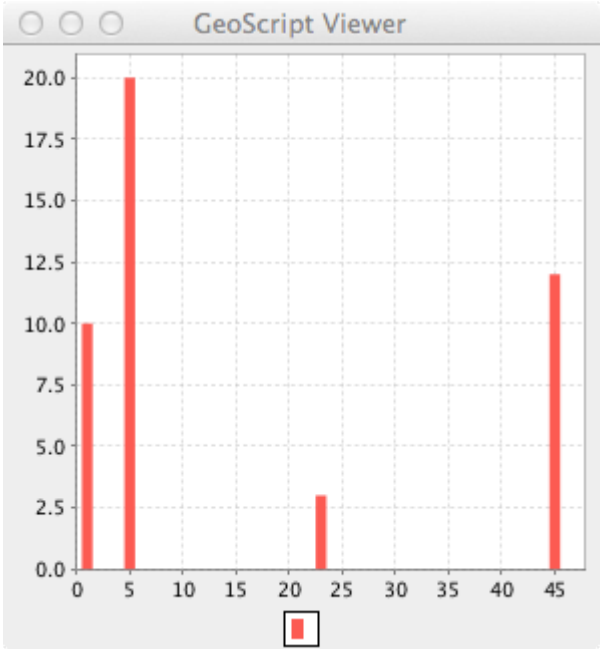


Plot Recipes

Processing Charts

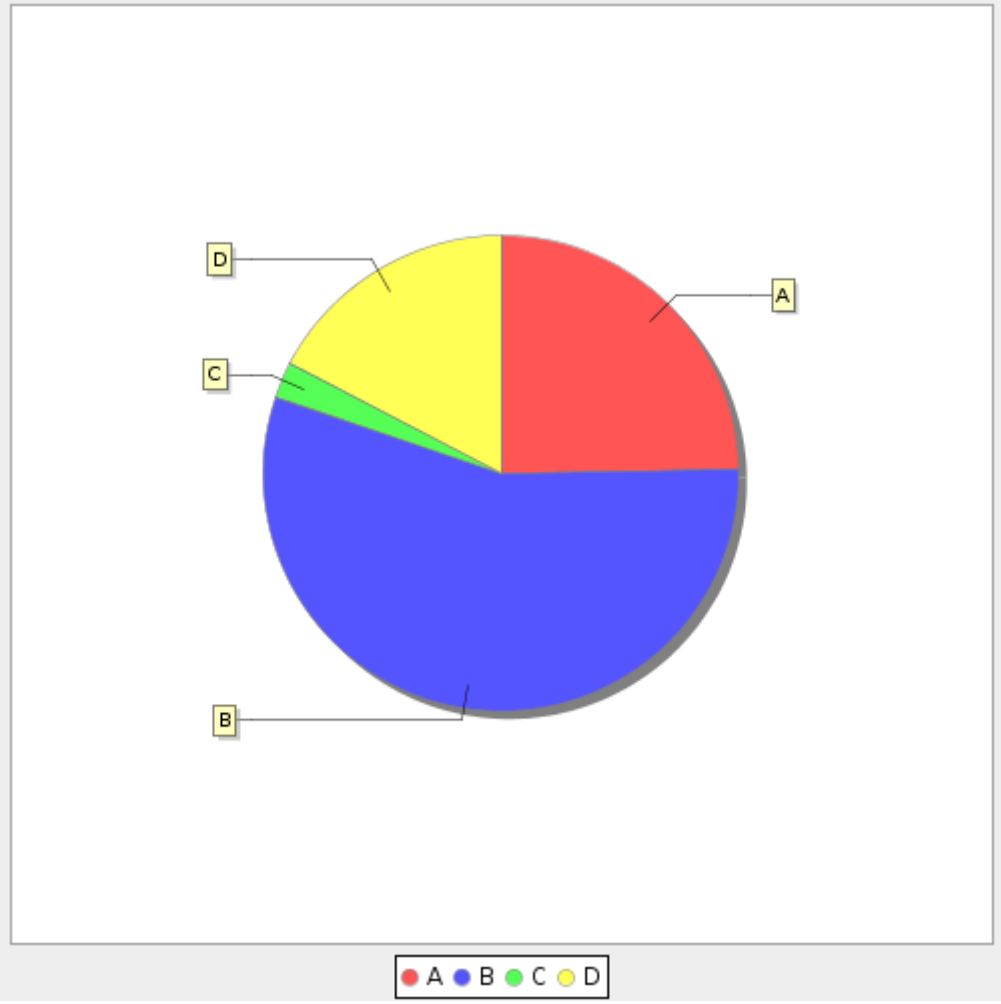
Show a chart in a GUI

```
List data = [
        [1,10],[45,12],[23,3],[5,20]
]
Chart chart = Bar.xy(data)
```



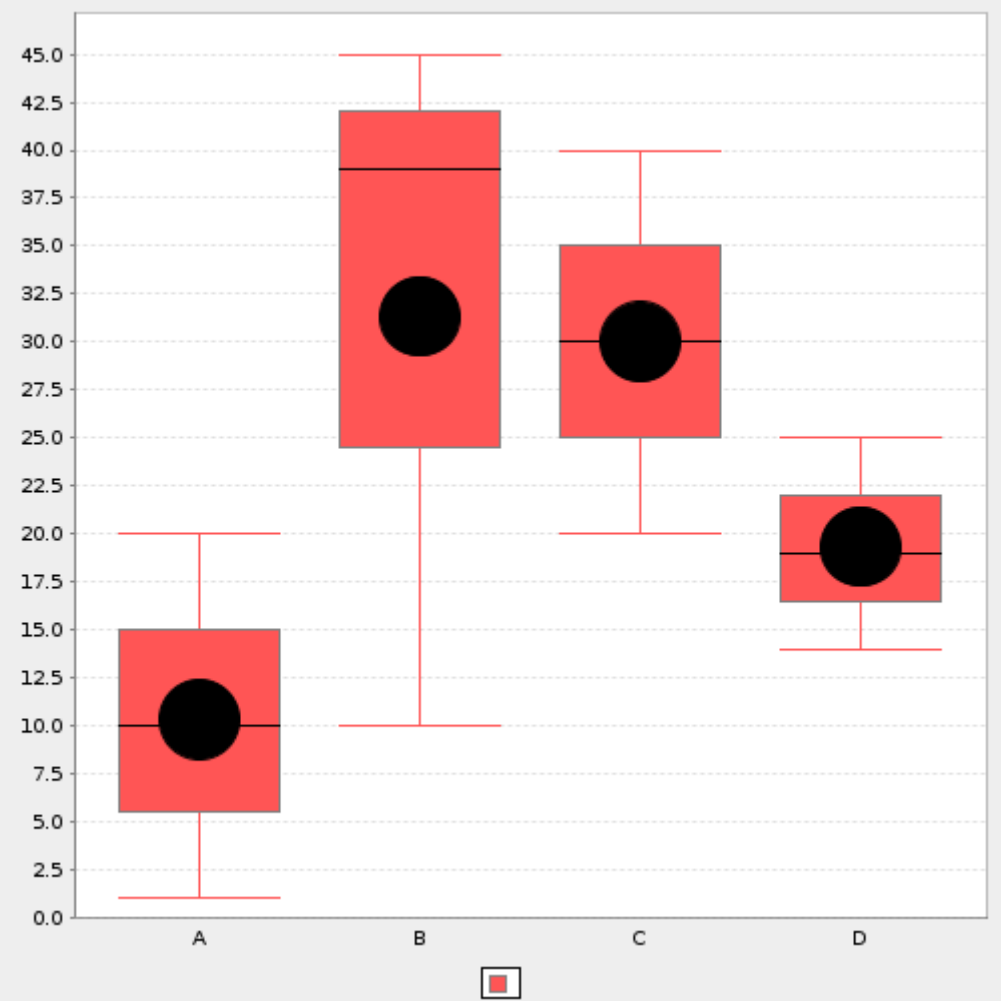
Get an image from a chart

```
Map data = [
        "A":20,"B":45,"C":2,"D":14
]
Chart chart = Pie.pie(data)
BufferedImage image = chart.image
```



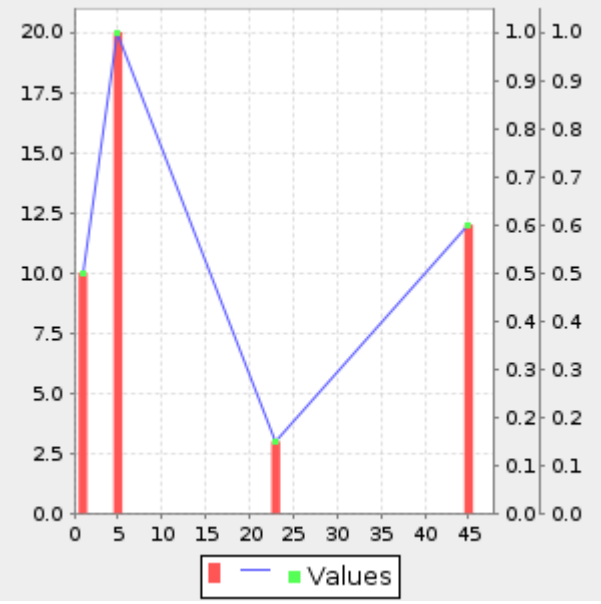
Save a chart to a file

```
Map data = [
        "A":[1,10,20],
        "B":[45,39,10],
        "C":[40,30,20],
        "D":[14,25,19]
]
Chart chart = Box.box(data)
File file = new File("chart.png")
chart.save(file)
```



Overlay multiple charts

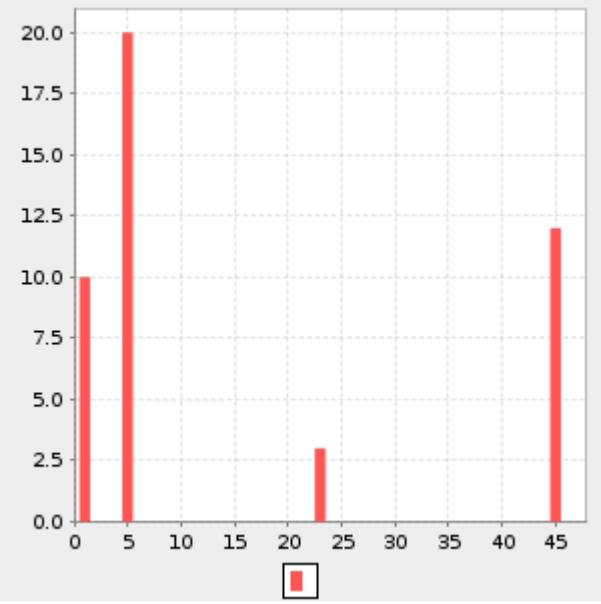
```
List data = [
        [1,10],[45,12],[23,3],[5,20]
]
Chart chart1 = Bar.xy(data)
Chart chart2 = Curve.curve(data)
Chart chart3 = Regression.linear(data)
chart1.overlay([chart2,chart3])
```



Creating Bar Charts

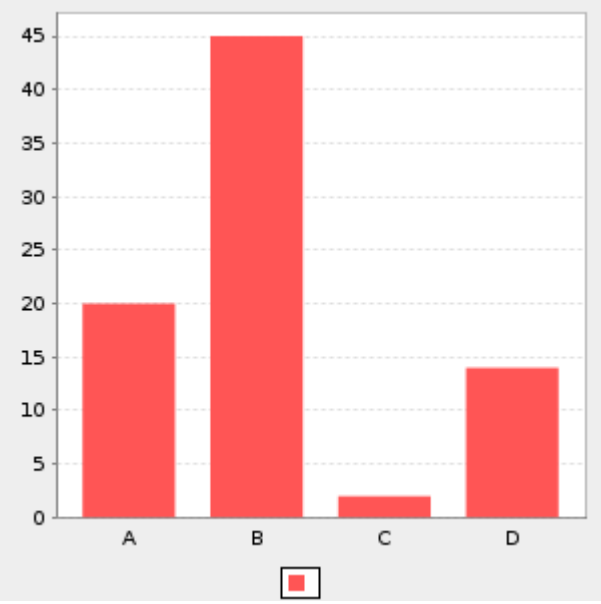
Create a basic bar chart

```
List data = [
        [1,10],[45,12],[23,3],[5,20]
]
Chart chart = Bar.xy(data)
```



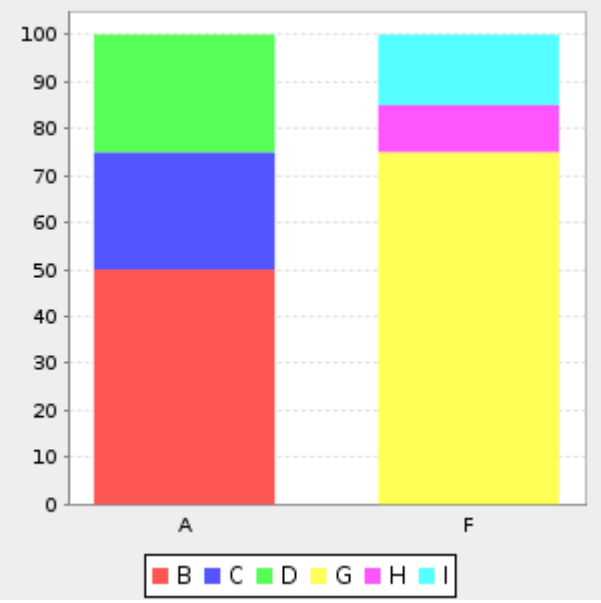
Create a bar chart with categories

```
Map data = [
        "A":20,"B":45,"C":2,"D":14
]
Chart chart = Bar.category(data)
```



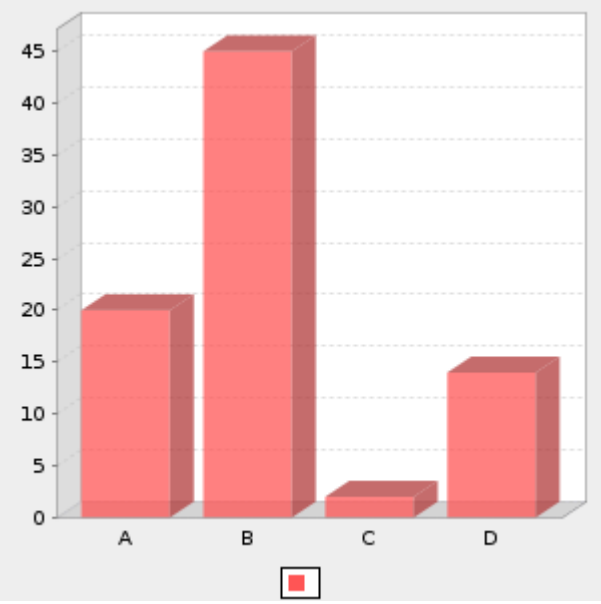
Create a stacked bar chart with two series of data

```
Map data = [
        "A": ["B":50,"C":25,"D":25],
        "F": ["G":75,"H":10,"I":15]
]
Chart chart = Bar.category(data, stacked: true)
```



Create a 3D bar chart with categories

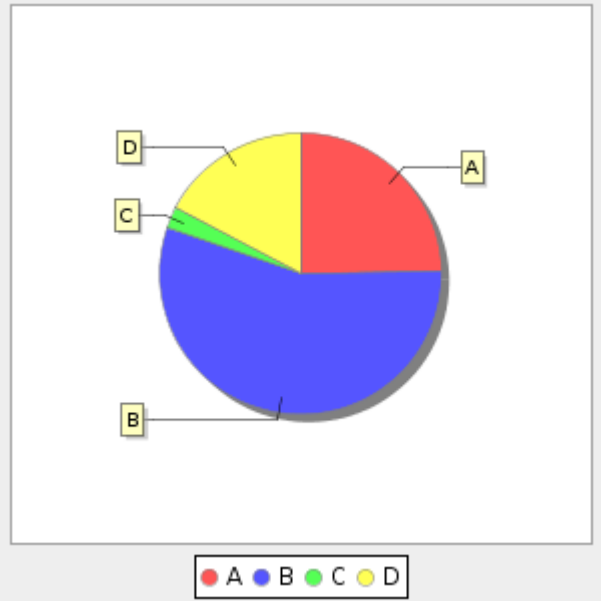
```
Map data = [
        "A":20,"B":45,"C":2,"D":14
]
Chart chart = Bar.category(data, trid: true)
```



Creating Pie Charts

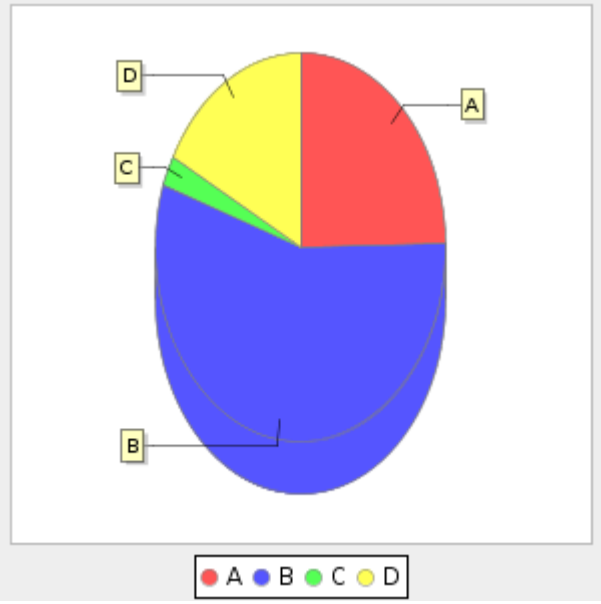
Create a pie chart

```
Map data = [
        "A":20,"B":45,"C":2,"D":14
]
Chart chart = Pie.pie(data)
```



Create a 3D pie chart

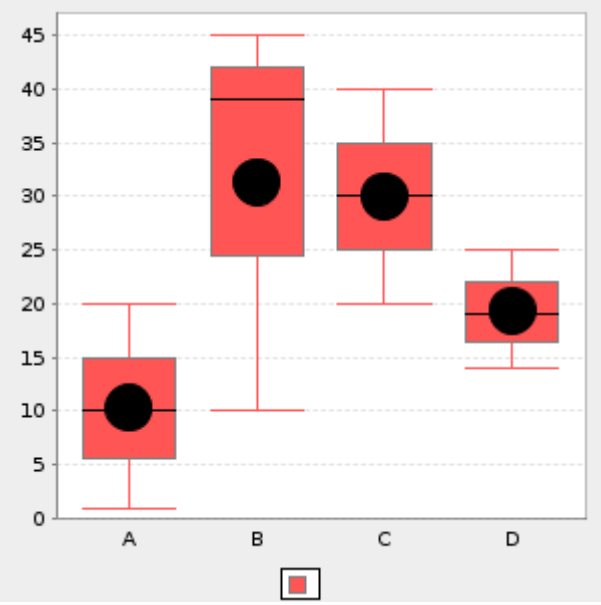
```
Map data = [
        "A":20,"B":45,"C":2,"D":14
]
Chart chart = Pie.pie(data, trid: true)
```



Creating Box Charts

Create a box chart

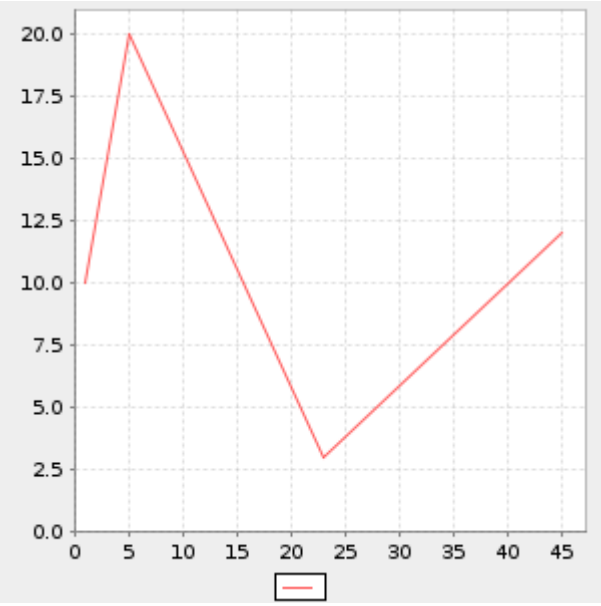
```
Map data = [
        "A":[1,10,20],
        "B":[45,39,10],
        "C":[40,30,20],
        "D":[14,25,19]
]
Chart chart = Box.box(data)
```



Creating Curve Charts

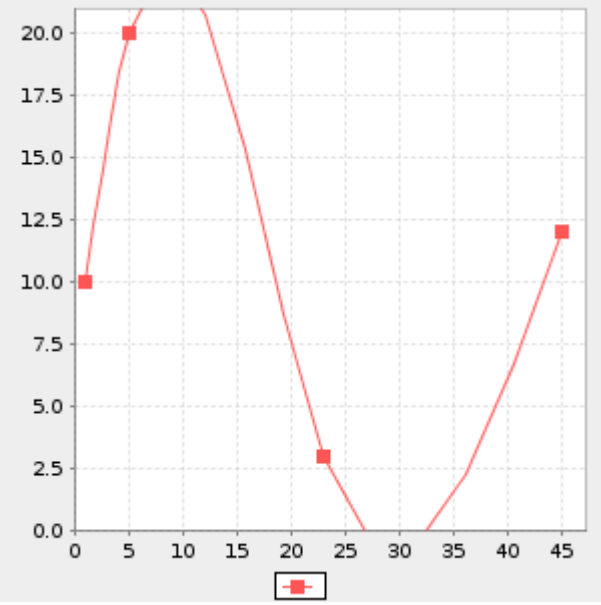
Create a curve chart

```
List data = [
        [1,10],[45,12],[23,3],[5,20]
]
Chart chart = Curve.curve(data)
```



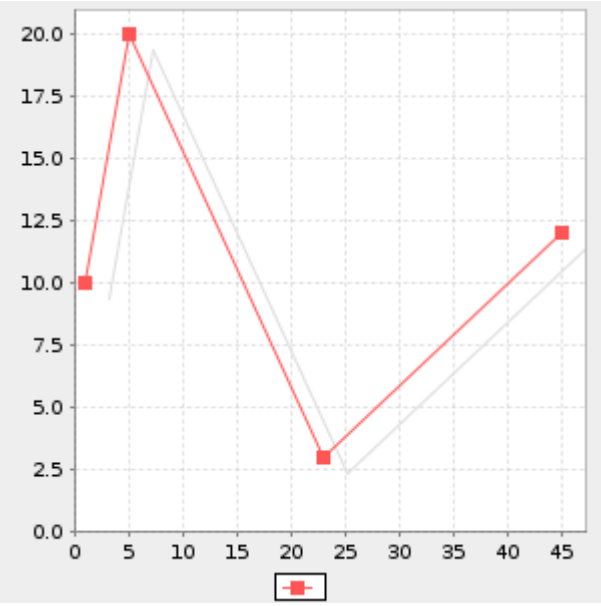
Create a smooth curve chart

```
List data = [
        [1,10],[45,12],[23,3],[5,20]
]
Chart chart = Curve.curve(data, smooth: true)
```



Create a 3D curve chart

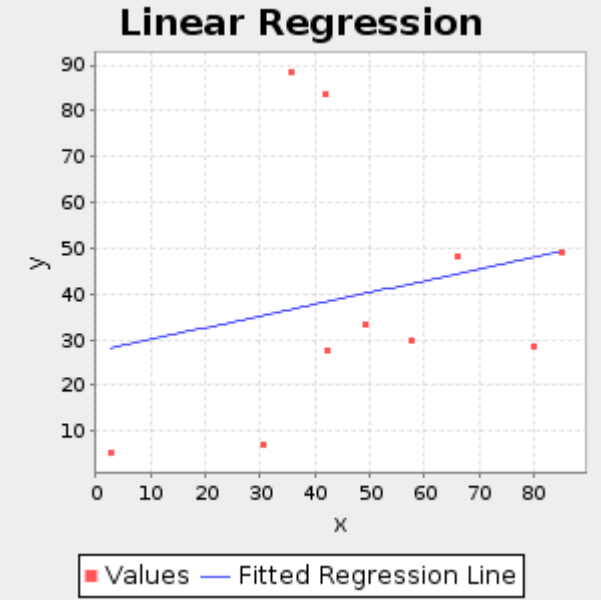
```
List data = [
        [1,10],[45,12],[23,3],[5,20]
]
Chart chart = Curve.curve(data, trid: true)
```



Creating Regression Charts

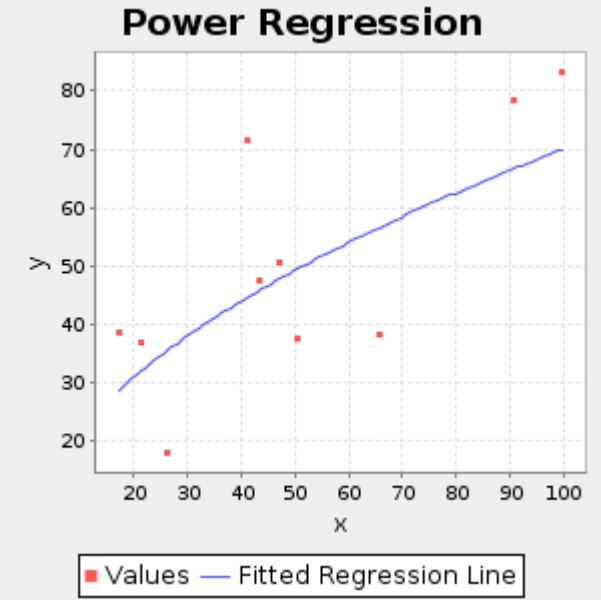
Create a linear regression chart

```
MultiPoint mulitPoint = Geometry.createRandomPoints(new Bounds(0,0,100,100).geometry,
10)
List data = mulitPoint.geometries.collect{ Point pt ->
    [pt.x, pt.y]
}
Chart chart = Regression.linear(data)
```



Create a power regression chart

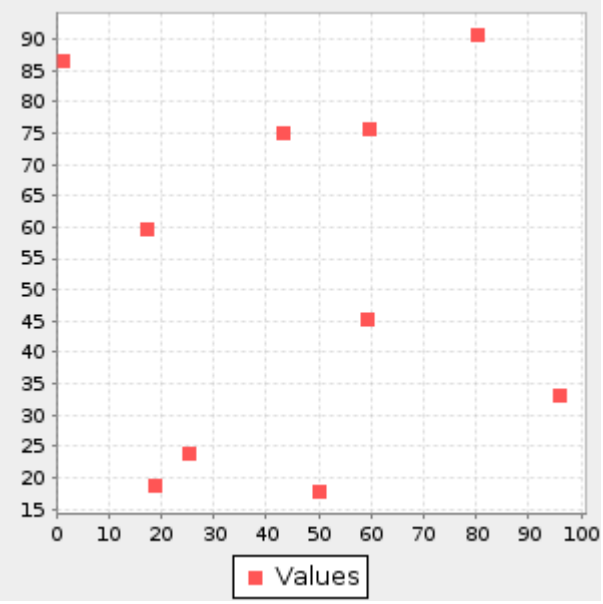
```
MultiPoint mulitPoint = Geometry.createRandomPoints(new Bounds(0,0,100,100).geometry,
10)
List data = mulitPoint.geometries.collect{ Point pt ->
    [pt.x, pt.y]
}
Chart chart = Regression.power(data)
```



Creating Scatter Plot Charts

Create a scatter plot chart

```
MultiPoint mulitPoint = Geometry.createRandomPoints(new Bounds(0,0,100,100).geometry,
10)
List data = mulitPoint.geometries.collect{ Point pt ->
    [pt.x, pt.y]
}
Chart chart = Scatter.scatterplot(data)
```



Feature Recipes

Creating Fields

Create a Field with a name and a type

```
Field field = new Field("name", "String")
println field
```

```
name: String
```

Create a Geometry Field with a name and a geometry type and an optional projection

```
Field field = new Field("geom", "Point", "EPSG:4326")
println field
```

```
geom: Point(EPSG:4326)
```

Create a Field with a List of Strings (name, type, projection)

```
Field field = new Field(["geom", "Polygon", "EPSG:4326"])
println field
```

```
geom: Polygon(EPSG:4326)
```

Create a Field from a Map where keys are name, type, proj

```
Field field = new Field([
        "name": "geom",
        "type": "LineString",
        "proj": new Projection("EPSG:4326")
])
println field
```

```
geom: LineString(EPSG:4326)
```

Access a Field's properties

```
Field field = new Field("geom", "Point", "EPSG:4326")
println "Name = ${field.name}"
println "Type = ${field.typ}"
println "Projection = ${field.proj}"
println "Is Geometry = ${field.geometry}"
```

```
Name = geom
Type = Point
Projection = "EPSG:4326
Is Geometry = true
```

Creating Schemas

Create a Schema from a list of Fields

```
Schema schema = new Schema("cities", [
        new Field("geom", "Point", "EPSG:4326"),
        new Field("id", "Integer"),
        new Field("name", "String")
])
println schema
```

```
cities geom: Point(EPSG:4326), id: Integer, name: String
```

Create a Schema from a list of Lists

```
Schema schema = new Schema("cities", [
        ["geom", "Point", "EPSG:4326"],
        ["id", "Integer"],
        ["name", "String"]
])
println schema
```

```
cities geom: Point(EPSG:4326), id: Integer, name: String
```

Create a Schema from a list of Maps

```
Schema schema = new Schema("cities", [
        [name: "geom", type: "Point", proj: "EPSG:4326"],
        [name: "id",   type: "Integer"],
        [name: "name", type: "String"]
])
println schema
```

```
cities geom: Point(EPSG:4326), id: Integer, name: String
```

Create a Schema from a string

```
Schema schema = new Schema("cities", "geom:Point:srid=4326,id:Integer,name:String")
println schema
```

```
cities geom: Point(EPSG:4326), id: Integer, name: String
```

Getting Schema Properties

Get the Schema's name

```
Schema schema = new Schema("cities", [
        new Field("geom", "Point", "EPSG:4326"),
        new Field("id", "Integer"),
        new Field("name", "String")
], "https://github.com/jericks/geoscript-groovy-cookbook")
String name = schema.name
println name
```

```
cities
```

Get the Schema's geometry Field

```
Field geomField = schema.geom
println geomField
```

```
geom: Point(EPSG:4326)
```

Get the Schema's Projection

```
Projection proj = schema.proj
println proj
```

```
EPSG:4326
```

Get the Schema's URI

```
String uri = schema.uri
println uri
```

```
https://github.com/jericks/geoscript-groovy-cookbook
```

Get the Schema's specification string

```
String spec = schema.spec
println spec
```

```
geom:Point:srid=4326,id:Integer,name:String
```

Getting Schema Fields

Get the Schema's Fields

```
Schema schema = new Schema("cities", [
        new Field("geom", "Point", "EPSG:4326"),
        new Field("id", "Integer"),
        new Field("name", "String")
])
List<Field> fields = schema.fields
fields.each { Field field ->
    println field
}
```

```
geom: Point(EPSG:4326)
id: Integer
name: String
```

Get a Field

```
Field nameField = schema.field("name")
println nameField
```

```
name: String
```

Get a Field

```
Field idField = schema.get("id")
println idField
```

```
id: Integer
```

Check if a Schema has a Field

```
boolean hasArea = schema.has("area")
println "Has area Field? ${hasArea}"

boolean hasGeom = schema.has("geom")
println "Has geom Field? ${hasGeom}"
```

```
false
true
```

Modifying Schemas

Change the projection of a Schema

```
Schema schema = new Schema("cities", [
        new Field("geom", "Point", "EPSG:4326"),
        new Field("id", "Integer"),
        new Field("name", "String")
])
Schema reprojectedSchema = schema.reproject("EPSG:2927", "cities_spws")
```

```
cities_spws geom: Point(EPSG:2927), id: Integer, name: String
```

Change the geometry type of a Schema

```
Schema schema = new Schema("cities", [
        new Field("geom", "Point", "EPSG:4326"),
        new Field("id", "Integer"),
        new Field("name", "String")
])
Schema polyognSchema = schema.changeGeometryType("Polygon", "cities_buffer")
```

```
cities_buffer geom: Polygon(EPSG:4326), id: Integer, name: String
```

Change a Field definition of a Schema

```
Schema schema = new Schema("cities", [
        new Field("geom", "Point", "EPSG:4326"),
        new Field("id", "Integer"),
        new Field("name", "String")
])
Schema guidSchema = schema.changeField(schema.field('id'), new Field('guid','String'),
'cities_guid')
```

```
cities_guid geom: Point(EPSG:4326), guid: String, name: String
```

Change Field definitions of a Schema

```
Schema schema = new Schema("cities", [
        new Field("geom", "Point", "EPSG:4326"),
        new Field("id", "Integer"),
        new Field("name", "String")
])
Schema updatedSchema = schema.changeFields(
        [
            (schema.field('id'))   : new Field('guid','String'),
            (schema.field('name')) : new Field('description','String')
        ], 'cities_updated')
```

```
cities_updated geom: Point(EPSG:4326), guid: String, description: String
```

Add a Field to a Schema

```
Schema schema = new Schema("countries", [
        new Field("geom", "Polygon", "EPSG:4326"),
        new Field("id", "Integer"),
        new Field("name", "String")
])
Schema updatedSchema = schema.addField(new Field("area", "Double"), "countries_area")
```

```
countries_area geom: Polygon(EPSG:4326), id: Integer, name: String, area: Double
```

Add a List of Fields to a Schema

```
Schema schema = new Schema("countries", [
        new Field("geom", "Polygon", "EPSG:4326"),
        new Field("id", "Integer"),
        new Field("name", "String")
])
Schema updatedSchema = schema.addFields([
        new Field("area", "Double"),
        new Field("perimeter", "Double"),
], "countries_areaperimeter")
```

```
countries_areaperimeter geom: Polygon(EPSG:4326), id: Integer, name: String, area:
Double, perimeter: Double
```

Remove a Field from a Schema

```
Schema schema = new Schema("countries", [
        new Field("geom", "Polygon", "EPSG:4326"),
        new Field("id", "Integer"),
        new Field("name", "String"),
        new Field("area", "Double")
])
Schema updatedSchema = schema.removeField(schema.field("area"), "countries_updated")
```

```
countries_updated geom: Polygon(EPSG:4326), id: Integer, name: String
```

Remove a List of Fields from a Schema

```
Schema schema = new Schema("countries", [
        new Field("geom", "Polygon", "EPSG:4326"),
        new Field("id", "Integer"),
        new Field("name", "String"),
        new Field("area", "Double")
])
Schema updatedSchema = schema.removeFields([
        schema.field("area"),
        schema.field("name")
], "countries_updated")
```

```
countries_updated geom: Polygon(EPSG:4326), id: Integer
```

Create a new Schema from an existing Schema but only including a subset of Fields

```
Schema schema = new Schema("countries", [
        new Field("geom", "Polygon", "EPSG:4326"),
        new Field("id", "Integer"),
        new Field("name", "String"),
        new Field("area", "Double")
])
Schema updatedSchema = schema.removeFields([
        schema.field("area"),
        schema.field("name")
], "countries_updated")
```

```
countries_updated geom: Polygon(EPSG:4326), name: String
```

Combining Schemas

Combining two Schemas results in a Map with two values: schema and fields. The schema property contains the new Schema. The fields property is List of two Maps which both contain a mapping between the fields of the original Schema and the newly created Schema.

Optional arguments to the Schema.addSchema method are:

- postfixAll: Whether to postfix all field names (true) or not (false). If true, all Fields from the this current Schema will have '1' at the end of their name while the other Schema's Fields will have '2'. Defaults to false.
- includeDuplicates: Whether or not to include duplicate fields names. Defaults to false. If a duplicate is found a '2' will be added.
- maxFieldNameLength: The maximum new Field name length (mostly to support shapefiles where Field names can't be longer than 10 characters

- firstPostfix: The postfix string (default is '1') for Fields from the current Schema. Only applicable when postfixAll or includeDuplicates is true.
- secondPostfix: The postfix string (default is '2') for Fields from the other Schema. Only applicable when postfixAll or includeDuplicates is true.

Combine two Schemas with no duplicate fields and no postfixes to field names

```
Schema shopSchema = new Schema("shops", [
        new Field("geom", "Point", "EPSG:4326"),
        new Field("id", "Integer"),
        new Field("name", "String")
])

Schema cafeSchema = new Schema("cafes", [
        new Field("geom", "Point", "EPSG:4326"),
        new Field("id", "Integer"),
        new Field("name", "String"),
        new Field("address", "String")
])

Map result = shopSchema.addSchema(cafeSchema, "business")

Schema combinedSchema = result.schema
println combinedSchema
```

```
business geom: Point(EPSG:4326), id: Integer, name: String, address: String
```

```
Map<String,String> shopSchemaFieldMapping = result.fields[0]
println shopSchemaFieldMapping
```

```
[geom:geom, id:id, name:name]
```

```
Map<String,String> cafeSchemaSchemaFieldMapping = result.fields[1]
println cafeSchemaSchemaFieldMapping
```

```
[address:address]
```

Combine two Schemas with no duplicate fields and postfixes

```
Schema shopSchema = new Schema("shops", [
        new Field("geom", "Point", "EPSG:4326"),
        new Field("id", "Integer"),
        new Field("name", "String")
])

Schema cafeSchema = new Schema("cafes", [
        new Field("geom", "Point", "EPSG:4326"),
        new Field("id", "Integer"),
        new Field("name", "String"),
        new Field("address", "String")
])

Map result = shopSchema.addSchema(cafeSchema, "business", postfixAll: true,
includeDuplicates: false)

Schema combinedSchema = result.schema
println combinedSchema
```

```
business geom: Point(EPSG:4326), id1: Integer, name1: String, id2: Integer, name2:
String, address2: String
```

```
Map<String,String> shopSchemaFieldMapping = result.fields[0]
println shopSchemaFieldMapping
```

```
[geom:geom, id:id1, name:name1]
```

```
Map<String,String> cafeSchemaSchemaFieldMapping = result.fields[1]
println cafeSchemaSchemaFieldMapping
```

```
[id:id2, name:name2, address:address2]
```

Creating Features from a Schema

Create a Feature from a Schema with a Map of values

```
Schema schema = new Schema("cities", [
        new Field("geom", "Point", "EPSG:4326"),
        new Field("id", "Integer"),
        new Field("name", "String")
])
Feature feature = schema.feature([
        id: 1,
        name: 'Seattle',
        geom: new Point( -122.3204, 47.6024)
], "city.1")
println feature
```

```
cities.city.1 geom: POINT (-122.3204 47.6024), id: 1, name: Seattle
```

Create a Feature from a Schema with a List of values. The order of the values must match the order of the Fields.

```
Schema schema = new Schema("cities", [
        new Field("geom", "Point", "EPSG:4326"),
        new Field("id", "Integer"),
        new Field("name", "String")
])
Feature feature = schema.feature([
        new Point( -122.3204, 47.6024),
        1,
        'Seattle'
], "city.1")
println feature
```

```
cities.city.1 geom: POINT (-122.3204 47.6024), id: 1, name: Seattle
```

Create a Feature from a Schema with another Feature.

```
Schema schema = new Schema("cities", [
        new Field("geom", "Point", "EPSG:4326"),
        new Field("id", "Integer"),
        new Field("name", "String")
])
Feature feature1 = new Feature([
        id: 1,
        name: 'Seattle',
        geom: new Point( -122.3204, 47.6024)
], "city.1", schema)
println feature1
Feature feature2 = schema.feature(feature1)
println feature2
```

```
cities.city.1 geom: POINT (-122.3204 47.6024), id: 1, name: Seattle
cities.city.1 geom: POINT (-122.3204 47.6024), id: 1, name: Seattle
```

Create an empty Feature from a Schema.

```
Schema schema = new Schema("cities", [
        new Field("geom", "Point", "EPSG:4326"),
        new Field("id", "Integer"),
        new Field("name", "String")
])
Feature feature = schema.feature()
println feature
```

```
cities.fid-7560f95d_15a53861eda_-7ffc geom: null, id: null, name: null
```

Reading and Writing Schemas

Finding Schema Writer and Readers

List all Schema Writers

```
List<SchemaWriter> writers = SchemaWriters.list()
writers.each { SchemaWriter writer ->
    println writer.class.simpleName
}
```

```
JsonSchemaWriter
StringSchemaWriter
XmlSchemaWriter
```

Find a Schema Writer

```
Schema schema = new Schema("cities", [
        new Field("geom", "Point", "EPSG:4326"),
        new Field("id", "Integer"),
        new Field("name", "String")
])

SchemaWriter writer = SchemaWriters.find("string")
String schemaStr = writer.write(schema)
println schemaStr
```

```
geom:Point:srid=4326,id:Integer,name:String
```

List all Schema Readers

```
List<SchemaReader> readers = SchemaReaders.list()
readers.each { SchemaReader reader ->
    println reader.class.simpleName
}
```

```
JsonSchemaReader
StringSchemaReader
XmlSchemaReader
```

Find a Schema Reader

```
SchemaReader reader = SchemaReaders.find("string")
Schema schema = reader.read("geom:Point:srid=4326,id:Integer,name:String")
println schema
```

```
layer geom: Point(EPSG:4326), id: Integer, name: String
```

String

Read a Schema from a String

```
StringSchemaReader reader = new StringSchemaReader()
Schema schema = reader.read("geom:Point:srid=4326,id:Integer,name:String", name:
"points")
println schema
```

```
points geom: Point(EPSG:4326), id: Integer, name: String
```

Write a Schema to a String

```
Schema schema = new Schema("cities", [
        new Field("geom", "Point", "EPSG:4326"),
        new Field("id", "Integer"),
        new Field("name", "String")
])

StringSchemaWriter writer = new StringSchemaWriter()
String schemaStr = writer.write(schema)
println schemaStr
```

```
geom:Point:srid=4326,id:Integer,name:String
```

JSON

Read a Schema from a JSON

```
        JsonSchemaReader reader = new JsonSchemaReader()
        Schema schema = reader.read("""{
    "name": "cities",
    "projection": "EPSG:4326",
    "geometry": "geom",
    "fields": [
        {
            "name": "geom",
            "type": "Point",
            "geometry": true,
            "projection": "EPSG:4326"
        },
        {
            "name": "id",
            "type": "Integer"
        },
        {
            "name": "name",
            "type": "String"
        }
    ]
}""")
        println schema
```

```
cities geom: Point(EPSG:4326), id: Integer, name: String
```

Write a Schema to a JSON

```
Schema schema = new Schema("cities", [
        new Field("geom", "Point", "EPSG:4326"),
        new Field("id", "Integer"),
        new Field("name", "String")
])

JsonSchemaWriter writer = new JsonSchemaWriter()
String schemaStr = writer.write(schema)
println schemaStr
```

```
{
    "name": "cities",
    "projection": "EPSG:4326",
    "geometry": "geom",
    "fields": [
        {
            "name": "geom",
            "type": "Point",
            "geometry": true,
            "projection": "EPSG:4326"
        },
        {
            "name": "id",
            "type": "Integer"
        },
        {
            "name": "name",
            "type": "String"
        }
    ]
}
```

XML

Read a Schema from a XML

```
        XmlSchemaReader reader = new XmlSchemaReader()
        Schema schema = reader.read("""<schema>
  <name>cities</name>
  <projection>EPSG:4326</projection>
  <geometry>geom</geometry>
  <fields>
    <field>
      <name>geom</name>
      <type>Point</type>
      <projection>EPSG:4326</projection>
    </field>
    <field>
      <name>id</name>
      <type>Integer</type>
    </field>
    <field>
      <name>name</name>
      <type>String</type>
    </field>
  </fields>
</schema>""")
        println schema
```

```
cities geom: Point(EPSG:4326), id: Integer, name: String
```

Write a Schema to a XML

```
Schema schema = new Schema("cities", [
        new Field("geom", "Point", "EPSG:4326"),
        new Field("id", "Integer"),
        new Field("name", "String")
])

XmlSchemaWriter writer = new XmlSchemaWriter()
String schemaStr = writer.write(schema)
println schemaStr
```

```
<schema>
  <name>cities</name>
  <projection>EPSG:4326</projection>
  <geometry>geom</geometry>
  <fields>
    <field>
      <name>geom</name>
      <type>Point</type>
      <projection>EPSG:4326</projection>
    </field>
    <field>
      <name>id</name>
      <type>Integer</type>
    </field>
    <field>
      <name>name</name>
      <type>String</type>
    </field>
  </fields>
</schema>
```

Creating Features

Create an empty Feature from a Map of values and a Schema.

```
Schema schema = new Schema("cities", [
        new Field("geom", "Point", "EPSG:4326"),
        new Field("id", "Integer"),
        new Field("name", "String")
])
Feature feature = new Feature([
        id: 1,
        name: "Seattle",
        geom: new Point(-122.3204, 47.6024)
], "city.1", schema)
println feature
```

```
cities.city.1 geom: POINT (-122.3204 47.6024), id: 1, name: Seattle
```

Create an empty Feature from a List of values and a Schema.

```
Schema schema = new Schema("cities", [
        new Field("geom", "Point", "EPSG:4326"),
        new Field("id", "Integer"),
        new Field("name", "String")
])
Feature feature = new Feature([
        new Point(-122.3204, 47.6024),
        1,
        "Seattle"
], "city.1", schema)
println feature
```

```
cities.city.1 geom: POINT (-122.3204 47.6024), id: 1, name: Seattle
```

Create an empty Feature from a Map of values. The Schema is inferred from the values.

```
Feature feature = new Feature([
  id: 1,
  name: "Seattle",
  geom: new Point(-122.3204, 47.6024)
], "city.1")
println feature
```

```
feature.city.1 id: 1, name: Seattle, geom: POINT (-122.3204 47.6024)
```

Getting Feature Properties

Get a Feature's ID

```
Schema schema = new Schema("cities", [
        new Field("geom", "Point", "EPSG:4326"),
        new Field("id", "Integer"),
        new Field("name", "String")
])
Feature feature = new Feature([
        new Point(-122.3204, 47.6024),
        1,
        "Seattle"
], "city.1", schema)

String id = feature.id
println id
```

```
city.1
```

Get a Feature's Geometry

```
Geometry geometry = feature.geom
println geometry
```

```
POINT (-122.3204 47.6024)
```

Get a Feature's Bounds

```
Bounds bounds = feature.bounds
println bounds
```

```
(-122.3204,47.6024,-122.3204,47.6024,EPSG:4326)
```

Get a Feature's attributes

```
Map attributes = feature.attributes
println attributes
```

```
[geom:POINT (-122.3204 47.6024), id:1, name:Seattle]
```

Getting Feature Attributes

Get an attribute from a Feature using a Field name

```
Schema schema = new Schema("cities", [
        new Field("geom", "Point", "EPSG:4326"),
        new Field("id", "Integer"),
        new Field("name", "String")
])
Feature feature = new Feature([
        new Point(-122.3204, 47.6024),
        1,
        "Seattle"
], "city.1", schema)

int id = feature.get("id")
println id
```

```
1
```

Get an attribute from a Feature using a Field

```
String name = feature.get(schema.field("name"))
println name
```

```
Seattle
```

Set an attribute of a Feature using a Field name and a new value

```
feature.set("name", "Tacoma")
println feature["name"]
```

```
Tacoma
```

Set an attribute of a Feature using a Field and a new value

```
feature.set(schema.field("name"), "Mercer Island")
println feature["name"]
```

```
Mercer Island
```

Set attributes of a Feature using a Map of new values

```
feature.set([id: 2])
println feature["id"]
```

```
2
```

Set a new Geometry value

```
feature.geom = new Point(-122.2220, 47.5673)
println feature.geom
```

```
POINT (-122.222 47.5673)
```

Reading and Writing Features

Finding Feature Writer and Readers

List all Feature Writers

```
List<Writer> writers = Writers.list()
writers.each { Writer writer ->
    println writer.class.simpleName
}
```

```
GeobufWriter
GeoJSONWriter
GeoRSSWriter
GmlWriter
GpxWriter
KmlWriter
```

Find a Feature Writer

```
Writer writer = Writers.find("geojson")
println writer.class.simpleName
```

```
GeoJSONWriter
```

List all Feature Readers

```
List<Reader> readers = Readers.list()
readers.each { Reader reader ->
    println reader.class.simpleName
}
```

```
GeobufReader
GeoJSONReader
GeoRSSReader
GmlReader
GpxReader
KmlReader
```

Find a Feature Reader

```
Reader reader = Readers.find("geojson")
println reader.class.simpleName
```

```
GeoJSONReader
```

GeoJSON

Get a GeoJSON String from a Feature

```
Schema schema = new Schema("cities", [
        new Field("geom", "Point", "EPSG:4326"),
        new Field("id", "Integer"),
        new Field("name", "String")
])
Feature feature = new Feature([
        new Point(-122.3204, 47.6024),
        1,
        "Seattle"
], "city.1", schema)

String geojson = feature.geoJSON
println geojson
```

```
{"type":"Feature","geometry":{"type":"Point","coordinates":[-
122.3204,47.6024]},"properties":{"id":1,"name":"Seattle"},"id":"city.1"}
```

Write a Feature to GeoJSON

```
Schema schema = new Schema("cities", [
        new Field("geom", "Point", "EPSG:4326"),
        new Field("id", "Integer"),
        new Field("name", "String")
])
Feature feature = new Feature([
        new Point(-122.3204, 47.6024),
        1,
        "Seattle"
], "city.1", schema)

GeoJSONWriter writer = new GeoJSONWriter()
String geojson = writer.write(feature)
println geojson
```

```
{"type":"Feature","geometry":{"type":"Point","coordinates":[-
122.3204,47.6024]},"properties":{"id":1,"name":"Seattle"},"id":"city.1"}
```

Get a Feature from GeoJSON

```
String geojson = '{"type":"Feature","geometry":{"type":"Point","coordinates":[-
122.3204,47.6024]},"properties":{"id":1,"name":"Seattle"},"id":"city.1"}'
Feature feature = Feature.fromGeoJSON(geojson)
println feature
```

```
feature.city.1 id: 1, name: Seattle, geometry: POINT (-122.3204 47.6024)
```

Read a Feature from GeoJSON

```
GeoJSONReader reader = new GeoJSONReader()
String geojson = '{"type":"Feature","geometry":{"type":"Point","coordinates":[-
122.3204,47.6024]},"properties":{"id":1,"name":"Seattle"},"id":"city.1"}'
Feature feature = reader.read(geojson)
println feature
```

```
feature.city.1 id: 1, name: Seattle, geometry: POINT (-122.3204 47.6024)
```

GeoBuf

Get a GeoBuf String from a Feature

```
Schema schema = new Schema("cities", [
        new Field("geom", "Point", "EPSG:4326"),
        new Field("id", "Integer"),
        new Field("name", "String")
])
Feature feature = new Feature([
        new Point(-122.3204, 47.6024),
        1,
        "Seattle"
], "city.1", schema)

String geobuf = feature.geobuf
println geobuf
```

```
0a0269640a046e616d65100218062a1d0a0c08001a089fd8d374c0ebb22d6a0218016a090a075365617474
6c65
```

Get a Feature from a GeoBuf String

```
String geobuf =
'0a0269640a046e616d65100218062a1d0a0c08001a089fd8d374c0ebb22d6a0218016a090a07536561747
46c65'
Feature feature = Feature.fromGeobuf(geobuf)
println feature
```

```
features.0 geom: POINT (-122.3204 47.6024), id: 1, name: Seattle
```

Write a Feature to a GeoBuf String

```
Schema schema = new Schema("cities", [
        new Field("geom", "Point", "EPSG:4326"),
        new Field("id", "Integer"),
        new Field("name", "String")
])
Feature feature = new Feature([
        new Point(-122.3204, 47.6024),
        1,
        "Seattle"
], "city.1", schema)

GeobufWriter writer = new GeobufWriter()
String geobuf = writer.write(feature)
println geobuf
```

```
0a0269640a046e616d65100218062a1d0a0c08001a089fd8d374c0ebb22d6a0218016a090a075365617474
6c65
```

Read a Feature from a GeoBuf String

```
GeobufReader reader = new GeobufReader()
String geobuf =
'0a0269640a046e616d65100218062a1d0a0c08001a089fd8d374c0ebb22d6a0218016a090a07536561747
46c65'
Feature feature = reader.read(geobuf)
println feature
```

```
features.0 geom: POINT (-122.3204 47.6024), id: 1, name: Seattle
```

GeoRSS

Get a GeoRSS String from a Feature

```
Schema schema = new Schema("cities", [
        new Field("geom", "Point", "EPSG:4326"),
        new Field("id", "Integer"),
        new Field("name", "String")
])
Feature feature = new Feature([
        new Point(-122.3204, 47.6024),
        1,
        "Seattle"
], "city.1", schema)

String georss = feature.geoRSS
println georss
```

```
<entry xmlns:georss='http://www.georss.org/georss'
xmlns='http://www.w3.org/2005/Atom'><title>city.1</title><summary>[geom:POINT (-
122.3204 47.6024), id:1, name:Seattle]</summary><updated>Sat Feb 18 23:19:43 UTC
2017</updated><georss:point>47.6024 -122.3204</georss:point></entry>
```

Get a Feature from a GeoRSS String

```
        String georss = """<entry xmlns:georss='http://www.georss.org/georss'
xmlns='http://www.w3.org/2005/Atom'>
    <title>city.1</title>
    <summary>[geom:POINT (-122.3204 47.6024), id:1, name:Seattle]</summary>
    <updated>Sat Jan 28 15:51:47 PST 2017</updated>
    <georss:point>47.6024 -122.3204</georss:point>
</entry>
"""
        Feature feature = Feature.fromGeoRSS(georss)
        println feature
```

```
georss.fid-7560f95d_15a53861eda_-8000 title: city.1, summary: [geom:POINT (-122.3204
47.6024), id:1, name:Seattle], updated: Sat Jan 28 15:51:47 PST 2017, geom: POINT (-
122.3204 47.6024)
```

Write a Feature to a GeoRSS String

```
Schema schema = new Schema("cities", [
        new Field("geom", "Point", "EPSG:4326"),
        new Field("id", "Integer"),
        new Field("name", "String")
])
Feature feature = new Feature([
        new Point(-122.3204, 47.6024),
        1,
        "Seattle"
], "city.1", schema)

GeoRSSWriter writer = new GeoRSSWriter()
String georss = writer.write(feature)
println georss
```

```
<entry xmlns:georss='http://www.georss.org/georss'
xmlns='http://www.w3.org/2005/Atom'><title>city.1</title><summary>[geom:POINT (-
122.3204 47.6024), id:1, name:Seattle]</summary><updated>Sat Feb 18 23:19:43 UTC
2017</updated><georss:point>47.6024 -122.3204</georss:point></entry>
```

Read a Feature from a GeoRSS String

```
        GeoRSSReader reader = new GeoRSSReader()
        String georss = """<entry xmlns:georss='http://www.georss.org/georss'
xmlns='http://www.w3.org/2005/Atom'>
    <title>city.1</title>
    <summary>[geom:POINT (-122.3204 47.6024), id:1, name:Seattle]</summary>
    <updated>Sat Jan 28 15:51:47 PST 2017</updated>
    <georss:point>47.6024 -122.3204</georss:point>
</entry>
"""
        Feature feature = reader.read(georss)
        println feature
```

```
georss.fid-7560f95d_15a53861eda_-7ffe title: city.1, summary: [geom:POINT (-122.3204
47.6024), id:1, name:Seattle], updated: Sat Jan 28 15:51:47 PST 2017, geom: POINT (-
122.3204 47.6024)
```

GML

Get a GML String from a Feature

```
Schema schema = new Schema("cities", [
        new Field("geom", "Point", "EPSG:4326"),
        new Field("id", "Integer"),
        new Field("name", "String")
])
Feature feature = new Feature([
        new Point(-122.3204, 47.6024),
        1,
        "Seattle"
], "city.1", schema)

String gml = feature.gml
println gml
```

```
<gsf:cities xmlns:gsf="http://geoscript.org/feature"
xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:gml="http://www.opengis.net/gml"
xmlns:xlink="http://www.w3.org/1999/xlink" fid="city.1">
<gml:name>Seattle</gml:name>
<gsf:geom>
<gml:Point>
<gml:coord>
<gml:X>-122.3204</gml:X>
<gml:Y>47.6024</gml:Y>
</gml:coord>
</gml:Point>
</gsf:geom>
<gsf:id>1</gsf:id>
</gsf:cities>
```

Get a Feature from a GML String

```
        String gml = """<gsf:cities xmlns:gsf="http://geoscript.org/feature"
xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:gml="http://www.opengis.net/gml"
xmlns:xlink="http://www.w3.org/1999/xlink" fid="city.1">
    <gml:name>Seattle</gml:name>
    <gsf:geom>
        <gml:Point>
            <gml:coord>
                <gml:X>-122.3204</gml:X>
                <gml:Y>47.6024</gml:Y>
            </gml:coord>
        </gml:Point>
    </gsf:geom>
    <gsf:id>1</gsf:id>
</gsf:cities>
"""
        Feature feature = Feature.fromGml(gml)
        println feature
```

```
feature.city.1 name: Seattle, id: 1, geom: POINT (-122.3204 47.6024)
```

Write a Feature to a GML String

```
Schema schema = new Schema("cities", [
        new Field("geom", "Point", "EPSG:4326"),
        new Field("id", "Integer"),
        new Field("name", "String")
])
Feature feature = new Feature([
        new Point(-122.3204, 47.6024),
        1,
        "Seattle"
], "city.1", schema)

GmlWriter writer = new GmlWriter()
String gml = writer.write(feature)
println gml
```

```
<gsf:cities xmlns:gsf="http://geoscript.org/feature"
xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:gml="http://www.opengis.net/gml"
xmlns:xlink="http://www.w3.org/1999/xlink" fid="city.1">
<gml:name>Seattle</gml:name>
<gsf:geom>
<gml:Point>
<gml:coord>
<gml:X>-122.3204</gml:X>
<gml:Y>47.6024</gml:Y>
</gml:coord>
</gml:Point>
</gsf:geom>
<gsf:id>1</gsf:id>
</gsf:cities>
```

Read a Feature from a GML String

```
        GmlReader reader = new GmlReader()
        String gml = """<gsf:cities xmlns:gsf="http://geoscript.org/feature"
xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:gml="http://www.opengis.net/gml"
xmlns:xlink="http://www.w3.org/1999/xlink" fid="city.1">
    <gml:name>Seattle</gml:name>
    <gsf:geom>
        <gml:Point>
            <gml:coord>
                <gml:X>-122.3204</gml:X>
                <gml:Y>47.6024</gml:Y>
            </gml:coord>
        </gml:Point>
    </gsf:geom>
    <gsf:id>1</gsf:id>
</gsf:cities>
"""
        Feature feature = reader.read(gml)
        println feature
```

```
feature.city.1 name: Seattle, id: 1, geom: POINT (-122.3204 47.6024)
```

GPX

Get a GPX String from a Feature

```
Schema schema = new Schema("cities", [
        new Field("geom", "Point", "EPSG:4326"),
        new Field("id", "Integer"),
        new Field("name", "String")
])
Feature feature = new Feature([
        new Point(-122.3204, 47.6024),
        1,
        "Seattle"
], "city.1", schema)

String gpx = feature.gpx
println gpx
```

```
<wpt lat='47.6024' lon='-122.3204'
xmlns='http://www.topografix.com/GPX/1/1'><name>city.1</name></wpt>
```

Get a Feature from a GPX String

```
String gpx = "<wpt lat='47.6024' lon='-122.3204'
xmlns='http://www.topografix.com/GPX/1/1'><name>city.1</name></wpt>"
Feature feature = Feature.fromGpx(gpx)
println feature
```

```
gpx.fid-7560f95d_15a53861eda_-7ffd geom: POINT (-122.3204 47.6024), name: city.1
```

Write a Feature to a GPX String

```
Schema schema = new Schema("cities", [
        new Field("geom", "Point", "EPSG:4326"),
        new Field("id", "Integer"),
        new Field("name", "String")
])
Feature feature = new Feature([
        new Point(-122.3204, 47.6024),
        1,
        "Seattle"
], "city.1", schema)

GpxWriter writer = new GpxWriter()
String gpx = writer.write(feature)
println gpx
```

```
<wpt lat='47.6024' lon='-122.3204'
xmlns='http://www.topografix.com/GPX/1/1'><name>city.1</name></wpt>
```

Read a Feature from a GPX String

```
GpxReader reader = new GpxReader()
String gpx = "<wpt lat='47.6024' lon='-122.3204'
xmlns='http://www.topografix.com/GPX/1/1'><name>city.1</name></wpt>"
Feature feature = reader.read(gpx)
println feature
```

```
gpx.fid-7560f95d_15a53861eda_-7fff geom: POINT (-122.3204 47.6024), name: city.1
```

KML

Get a KML String from a Feature

```
Schema schema = new Schema("cities", [
        new Field("geom", "Point", "EPSG:4326"),
        new Field("id", "Integer"),
        new Field("name", "String")
])
Feature feature = new Feature([
        new Point(-122.3204, 47.6024),
        1,
        "Seattle"
], "city.1", schema)

String kml = feature.kml
println kml
```

```
<kml:Placemark xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:kml="http://earth.google.com/kml/2.1" id="city.1">
<kml:name>Seattle</kml:name>
<kml:Point>
<kml:coordinates>-122.3204,47.6024</kml:coordinates>
</kml:Point>
</kml:Placemark>
```

Get a Feature from a KML String

```
        String kml = """<kml:Placemark xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:kml="http://earth.google.com/kml/2.1" id="city.1">
    <kml:name>Seattle</kml:name>
    <kml:Point>
        <kml:coordinates>-122.3204,47.6024</kml:coordinates>
    </kml:Point>
</kml:Placemark>"""
        Feature feature = Feature.fromKml(kml)
        println feature
```

```
placemark.city.1 name: Seattle, description: null, Geometry: POINT (-122.3204 47.6024)
```

Write a Feature to a KML String

```
Schema schema = new Schema("cities", [
        new Field("geom", "Point", "EPSG:4326"),
        new Field("id", "Integer"),
        new Field("name", "String")
])
Feature feature = new Feature([
        new Point(-122.3204, 47.6024),
        1,
        "Seattle"
], "city.1", schema)

KmlWriter writer = new KmlWriter()
String kml = writer.write(feature)
println kml
```

```
<kml:Placemark xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:kml="http://earth.google.com/kml/2.1" id="city.1">
<kml:name>Seattle</kml:name>
<kml:Point>
<kml:coordinates>-122.3204,47.6024</kml:coordinates>
</kml:Point>
</kml:Placemark>
```

Read a Feature from a KML String

```
        KmlReader reader = new KmlReader()
        String kml = """<kml:Placemark xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:kml="http://earth.google.com/kml/2.1" id="city.1">
    <kml:name>Seattle</kml:name>
    <kml:Point>
        <kml:coordinates>-122.3204,47.6024</kml:coordinates>
    </kml:Point>
</kml:Placemark>"""
        Feature feature = reader.read(kml)
        println feature
```

```
placemark.city.1 name: Seattle, description: null, Geometry: POINT (-122.3204 47.6024)
```

Filter Recipes

Creating Filters

Create a Filter from a CQL string

```
Filter filter = new Filter("name='Seattle'")
println filter.toString()
```

```
[ name = Seattle ]
```

Create a Filter from a CQL string

```
Filter filter = new Filter
("<filter><PropertyIsEqualTo><PropertyName>soilType</PropertyName><Literal>Mollisol</L
iteral></PropertyIsEqualTo></filter>")
println filter.toString()
```

```
[ soilType = Mollisol ]
```

Create a pass Filter that return true for everything

```
Filter filter = Filter.PASS
println filter.toString()
```

```
Filter.INCLUDE
```

Create a fail Filter that return false for everything

```
Filter filter = Filter.FAIL
println filter.toString()
```

```
Filter.EXCLUDE
```

Create a spatial bounding box Filter from a Bounds

```
Filter filter = Filter.bbox(new Bounds(-102, 43.5, -100, 47.5))
println filter.toString()
```

```
[ the_geom bbox POLYGON ((-102 43.5, -102 47.5, -100 47.5, -100 43.5, -102 43.5)) ]
```

Create a spatial contains Filter from a Geometry

```
Filter filter = Filter.contains(Geometry.fromWKT("POLYGON ((-104 45, -95 45, -95 50,
-104 50, -104 45))"))
println filter.toString()
```

```
[ the_geom contains POLYGON ((-104 45, -95 45, -95 50, -104 50, -104 45)) ]
```

Create a spatial distance within Filter from a Geometry and a distance

```
Filter filter = Filter.dwithin("the_geom", Geometry.fromWKT("POINT (-100 47)"), 10.2,
"feet")
println filter.toString()
```

```
[ the_geom dwithin POINT (-100 47), distance: 10.2 ]
```

Create a spatial crosses Filter from a Geometry

```
Filter filter = Filter.crosses("the_geom", Geometry.fromWKT("LINESTRING (-104 45, -95
45)"))
println filter.toString()
```

```
[ the_geom crosses LINESTRING (-104 45, -95 45) ]
```

Create a spatial intersects Filter from a Geometry

```
Filter filter = Filter.intersects(Geometry.fromWKT("POLYGON ((-104 45, -95 45, -95 50,
-104 50, -104 45))"))
println filter.toString()
```

```
[ the_geom intersects POLYGON ((-104 45, -95 45, -95 50, -104 50, -104 45)) ]
```

Create a feature id Filter

```
Filter filter = Filter.id("points.1")
println filter.toString()
```

```
[ points.1 ]
```

Create a feature ids Filter

```
Filter filter = Filter.ids(["points.1","points.2","points.3"])
println filter.toString()
```

```
[ points.1, points.2, points.3 ]
```

Using Filters

Get a CQL string from a Filter

```
Filter filter = new Filter("name='Seattle'")
String cql = filter.cql
println cql
```

```
name = 'Seattle'
```

Get an XML string from a Filter

```
String xml = filter.xml
println xml
```

```
<ogc:Filter xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:gml="http://www.opengis.net/gml" xmlns:ogc="http://www.opengis.net/ogc">
<ogc:PropertyIsEqualTo>
<ogc:PropertyName>name</ogc:PropertyName>
<ogc:Literal>Seattle</ogc:Literal>
</ogc:PropertyIsEqualTo>
</ogc:Filter>
```

Combine Filters with and

```
Filter cityFilter = new Filter("city = 'Seattle'")
Filter stateFilter = new Filter("state = 'WA'")
Filter andFilter = cityFilter.and(stateFilter)
println andFilter
```

```
[[ city = Seattle ] AND [ state = WA ]]
```

Combine Filters with and using the plus operator

```
Filter cityFilter = new Filter("city = 'Seattle'")
Filter stateFilter = new Filter("state = 'WA'")
Filter andFilter = cityFilter + stateFilter
println andFilter
```

```
[[ city = Seattle ] AND [ state = WA ]]
```

Combine Filters with or

```
Filter seattleFilter = new Filter("city = 'Seattle'")
Filter tacomaFilter = new Filter("city = 'Tacoma'")
Filter orFilter = seattleFilter.or(tacomaFilter)
println orFilter
```

```
[[ city = Seattle ] OR [ city = Tacoma ]]
```

Negate a Filter

```
Filter seattleFilter = new Filter("city = 'Seattle'")
Filter notSeattleFilter = seattleFilter.not
println notSeattleFilter
```

```
[ NOT [ city = Seattle ] ]
```

Negate a Filter using the minus operator

```
Filter seattleFilter = new Filter("city = 'Seattle'")
Filter notSeattleFilter = -seattleFilter
println notSeattleFilter
```

```
[ NOT [ city = Seattle ] ]
```

Simplify a Filter

```
Filter seattleFilter = new Filter("city = 'Seattle'")
Filter filter = (seattleFilter + Filter.PASS).simplify()
println filter
```

```
[ city = Seattle ]
```

Evaluating Filters

Test to see if a Filter matches a Feature by attribute

```
Feature feature = new Feature([
        id: 1,
        name: "Seattle",
        geom: new Point(-122.3204, 47.6024)
], "city.1")

Filter isNameFilter = new Filter("name='Seattle'")
boolean isName = isNameFilter.evaluate(feature)
println isName
```

```
true
```

```
Filter isNotNameFilter = new Filter("name='Tacoma'")
boolean isNotName = isNotNameFilter.evaluate(feature)
println isNotName
```

```
false
```

Test to see if a Filter matches a Feature by feature id

```
Filter isIdFilter = Filter.id("city.1")
boolean isId = isIdFilter.evaluate(feature)
println isId
```

```
true
```

```
Filter isNotIdFilter = Filter.id("city.2")
boolean isNotId = isNotIdFilter.evaluate(feature)
println isNotId
```

```
false
```

Test to see if a Filter matches a Feature by a spatial bounding box

```
Filter isInBboxFilter = Filter.bbox("geom",  new Bounds(-132.539, 42.811, -111.796,
52.268))
boolean isInBbox = isInBboxFilter.evaluate(feature)
println isInBbox
```

```
true
```

```
Filter isNotInBboxFilter = Filter.bbox("geom", new Bounds(-12.656, 18.979, 5.273,
34.597))
boolean isNotInBbox = isNotInBboxFilter.evaluate(feature)
println isNotInBbox
```

```
false
```

Creating Literals

Create a literal Expression from a number

```
Expression expression = new Expression(3.56)
println expression
```

```
3.56
```

Create a literal Expression from a string

```
Expression expression = new Expression("Seattle")
println expression
```

```
Seattle
```

Evaluating a literal Expression just gives you the value

```
Expression expression = new Expression(3.56)
double number = expression.evaluate()
println number
```

```
3.56
```

Creating Properties

Create a Property from a string

```
Property property = new Property("name")
println property
```

```
name
```

Create a Property from a Field

```
Field field = new Field("geom", "Polygon")
Property property = new Property(field)
println property
```

```
geom
```

Evaluating Properties

Evaluate a Property to get values from a Feature. Get the id

```
Feature feature = new Feature([
    id: 1,
    name: "Seattle",
    geom: new Point(-122.3204, 47.6024)
], "city.1")

Property idProperty = new Property("id")
int id = idProperty.evaluate(feature)
println id
```

```
1
```

Get the name

```
Property nameProperty = new Property("name")
String name = nameProperty.evaluate(feature)
println name
```

```
Seattle
```

Get the geometry

```
Property geomProperty = new Property("geom")
Geometry geometry = geomProperty.evaluate(feature)
println geometry
```

```
POINT (-122.3204 47.6024)
```

Creating Functions

Create a Function from a CQL string

```
Function function = new Function("centroid(the_geom)")
println function
```

```
centroid([the_geom])
```

Create a Function from a name and Expressions

```
Function function = new Function("centroid", new Property("the_geom"))
println function
```

```
centroid([the_geom])
```

Create a Function from a name, a Closure, and Expressions

```
Function function = new Function("my_centroid", {g-> g.centroid}, new Property
("the_geom"))
println function
```

```
my_centroid([the_geom])
```

Create a Function from a CQL string and a Closure

```
Function function = new Function("my_centroid(the_geom)", {g-> g.centroid})
println function
```

```
my_centroid([the_geom])
```

You can get a list of built in Functions

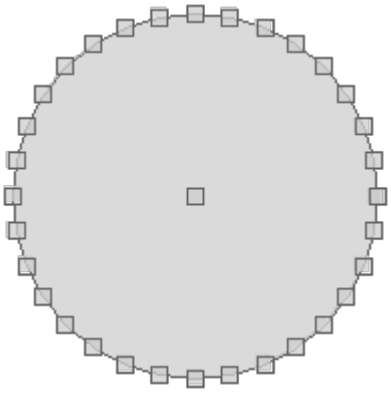
```
List<String> functionNames = Function.getFunctionNames()
println "There are ${functionNames.size()} Functions:"
functionNames.sort().subList(0,10).each { String name ->
    println name
}
```

```
There are 277 Functions:
Area
Categorize
Collection_Average
Collection_Bounds
Collection_Count
Collection_Max
Collection_Median
Collection_Min
Collection_Nearest
Collection_Sum
```

Evaluating Functions

Evaulate a geometry Function

```
Feature feature = new Feature([
        id: 1,
        name: "Seattle",
        geom: new Point(-122.3204, 47.6024)
], "city.1")

Function bufferFunction = new Function("buffer(geom, 10)")
Geometry polygon = bufferFunction.evaluate(feature)
```



Evaulate a geometry Function

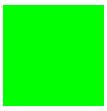
```
Function lowerCaseFunction = new Function("strToLowerCase(name)")
String lowerCaseName = lowerCaseFunction.evaluate(feature)
println lowerCaseName
```

```
seattle
```

Creating Colors

Create a Color from a RGB color string

```
Color color = new Color("0,255,0")
```



Create a Color from a CSS color name

```
Color color = new Color("silver")
```

Create a Color from a hexadecimal string

```
Color color = new Color("#0000ff")
```

Create a Color from a RGB List

```
Color color = new Color([255,0,0])
```

Create a Color from a RGB Map

```
Color color = new Color([r: 5, g: 35, b:45])
```

Create a Color from a HLS Map

```
Color color = new Color([h: 0, s: 1.0, l: 0.5])
```

Get a Random Color

```
Color color = Color.getRandom()
```

Get a Random Pastel Color

```
Color color = Color.getRandomPastel()
```

Get a darker Color

```
Color color = new Color("lightblue")
Color darkerColor = color.darker()
```

Get a brighter Color

```
Color color = new Color("purple")
Color brigtherColor = color.brighter()
```

Getting Color Formats

Create a Color

```
Color color = new Color("wheat")
```


Get Hex

```
String hex = color.hex
println hex
```

```
#f5deb3
```

Get RGB

```
List rgb = color.rgb
println rgb
```

```
[245, 222, 179]
```

Get HSL

```
List hsl = color.hsl
println hsl
```

```
[0.10858585256755147, 0.7674419030001307, 0.8313725489999999]
```

Get the java.awt.Color

```
java.awt.Color awtColor = color.asColor()
println awtColor
```

```
java.awt.Color[r=245,g=222,b=179]
```

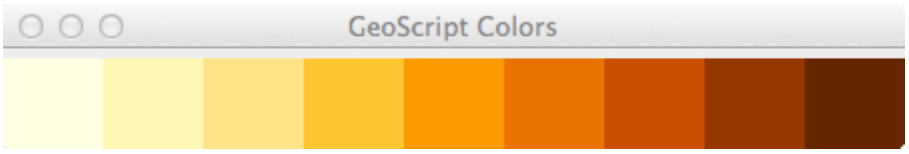
Displaying Colors

Draw a List of Colors to a BufferedImage

```
Color color = new Color("pink")
BufferedImage image = Color.drawToImage(
        [color.brighter(), color, color.darker()],
        "vertical",
        40
)
```

Draw a List of Colors to a simple GUI

```
List<Color> colors = Color.getPaletteColors("YlOrBr")
Color.draw(colors, "horizontal", 50)
```



Using Color Palettes

Get all color palettes

```
List<String> allPalettes = Color.getPaletteNames("all")
allPalettes.each { String name ->
    println name
}
```

```
YlOrRd
PRGn
PuOr
RdGy
Spectral
Grays
PuBuGn
RdPu
BuPu
YlOrBr
Greens
BuGn
Accents
GnBu
PuRd
Purples
RdYlGn
Paired
Blues
RdBu
Oranges
RdYlBu
PuBu
OrRd
Set3
Set2
Set1
Reds
PiYG
Dark2
YlGn
BrBG
YlGnBu
Pastel2
Pastel1
BlueToOrange
GreenToOrange
BlueToRed
GreenToRedOrange
Sunset
Green
YellowToRedHeatMap
BlueToYellowToRedHeatMap
DarkRedToYellowWhiteHeatMap
LightPurpleToDarkPurpleHeatMap
BoldLandUse
MutedTerrain
BoldLandUse
MutedTerrain
```

Get diverging color palettes

```
List<String> divergingPalettes = Color.getPaletteNames("diverging")
divergingPalettes.each { String name ->
    println name
}
```

```
PRGn
PuOr
RdGy
Spectral
RdYlGn
RdBu
RdYlBu
PiYG
BrBG
BlueToOrange
GreenToOrange
BlueToRed
GreenToRedOrange
```

Get sequential color palettes

```
List<String> sequentialPalettes = Color.getPaletteNames("sequential")
sequentialPalettes.each { String name ->
    println name
}
```

```
YlOrRd
Grays
PuBuGn
RdPu
BuPu
YlOrBr
Greens
BuGn
GnBu
PuRd
Purples
Blues
Oranges
PuBu
OrRd
Reds
YlGn
YlGnBu
Sunset
Green
YellowToRedHeatMap
BlueToYellowToRedHeatMap
DarkRedToYellowWhiteHeatMap
LightPurpleToDarkPurpleHeatMap
BoldLandUse
MutedTerrain
```

Get qualitative color palettes

```
List<String> qualitativePalettes = Color.getPaletteNames("qualitative")
qualitativePalettes.each { String name ->
    println name
}
```

```
Accents
Paired
Set3
Set2
Set1
Dark2
Pastel2
Pastel1
BoldLandUse
MutedTerrain
```

Get a Blue Green Color Palette

```
List colors = Color.getPaletteColors("BuGn")
```

Get a Purple Color Palette with only four colors

```
colors = Color.getPaletteColors("Purples", 4)
```

Get a Blue Green Color Palette

```
colors = Color.getPaletteColors("MutedTerrain")
```

Get a Blue Green Color Palette

```
colors = Color.getPaletteColors("BlueToYellowToRedHeatMap")
```

Create a Color palette by interpolating between two colors

```
Color startColor = new Color("red")
Color endColor = new Color("green")
List<Color> colors = startColor.interpolate(endColor, 10)
```

Create a Color palette by interpolating between two colors

```
Color startColor = new Color("wheat")
Color endColor = new Color("lightblue")
List<Color> colors = Color.interpolate(startColor, endColor, 8)
```

Creating Expressions from CQL

Create a literal number Expression from a CQL String

```
Expression expression = Expression.fromCQL("12")
println expression
```

```
12
```

Create a literal string Expression from a CQL String

```
Expression expression = Expression.fromCQL("'Washington'")
println expression
```

```
Washington
```

Create a Property from a CQL String

```
Property property = Expression.fromCQL("NAME")
println property
```

```
NAME
```

Create a Function from a CQL String

```
Function function = Expression.fromCQL("centroid(the_geom)")
println function
```

```
centroid([the_geom])
```

Process Recipes

Execute a built-in Process

Create a Process from a built-in process by name

```
Process process = new Process("vec:Bounds")
String name = process.name
println name
```

```
vec:Bounds
```

Get the title

```
String title = process.title
println title
```

```
Bounds
```

Get the description

```
String description = process.description
println description
```

```
Computes the bounding box of the input features.
```

Get the version

```
String version = process.version
println version
```

```
1.0.0
```

Get the input parameters

```
Map parameters = process.parameters
println parameters
```

```
[features:class geoscript.layer.Cursor]
```

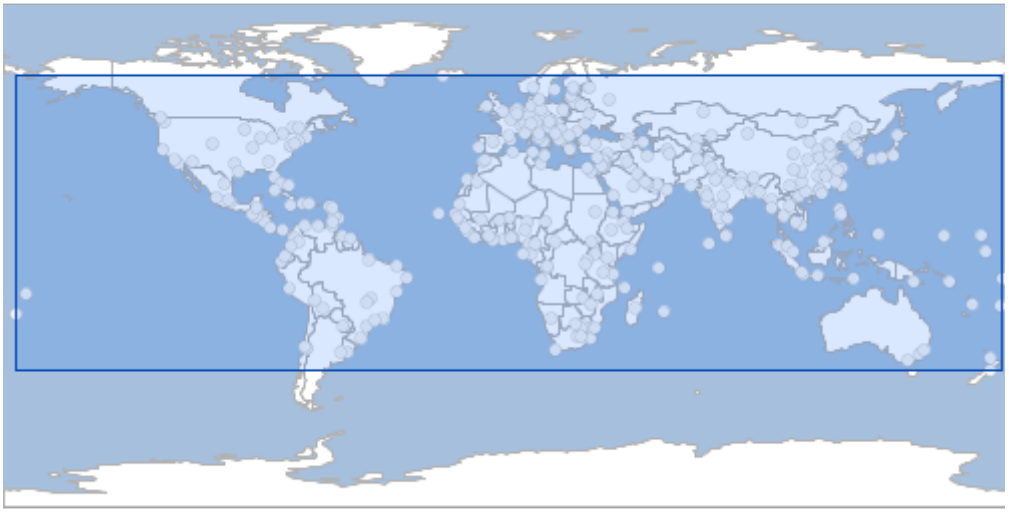
Get the output parameters

```
Map results = process.results
println results
```

```
[bounds:class geoscript.geom.Bounds]
```

Execute the Process to calculate the bounding box of all Features in a Layer

```
Workspace workspace = new GeoPackage('src/main/resources/data.gpkg')
Layer layer = workspace.get("places")
Map executeResults = process.execute([features: layer])
Bounds bounds = executeResults.bounds
```



Listing built-in Processes

Get the names of all built-in Processes

```
List<String> processes = Process.processNames
processes.each { String name ->
    println name
}
```

```
ras:AddCoverages
ras:Affine
ras:AreaGrid
ras:BandMerge
ras:BandSelect
ras:Contour
ras:ConvolveCoverage
ras:CoverageClassStats
ras:CropCoverage
ras:MultiplyCoverages
ras:NormalizeCoverage
ras:PolygonExtraction
ras:RangeLookup
ras:RasterAsPointCollection
ras:RasterZonalStatistics
ras:RasterZonalStatistics2
```

```
ras:ScaleCoverage
ras:StyleCoverage
vec:Aggregate
vec:BarnesSurface
vec:Bounds
vec:BufferFeatureCollection
vec:Centroid
vec:Clip
vec:CollectGeometries
vec:Count
vec:Feature
vec:FeatureClassStats
vec:Grid
vec:Heatmap
vec:InclusionFeatureCollection
vec:IntersectionFeatureCollection
vec:LRSGeocode
vec:LRSMeasure
vec:LRSSegment
vec:Nearest
vec:PointBuffers
vec:PointStacker
vec:Query
vec:RectangularClip
vec:Reproject
vec:Simplify
vec:Snap
vec:Transform
vec:UnionFeatureCollection
vec:Unique
vec:VectorToRaster
vec:VectorZonalStatistics
geo:buffer
geo:union
geo:intersection
geo:isValid
geo:difference
geo:distance
geo:area
geo:numGeometries
geo:isClosed
geo:crosses
geo:getGeometryN
geo:isSimple
geo:isWithinDistance
geo:overlaps
geo:relate
geo:symDifference
geo:touches
geo:within
geo:simplify
```

```
geo:densify
geo:reproject
geo:numPoints
geo:convexHull
geo:boundary
geo:centroid
geo:dimension
geo:exteriorRing
geo:numInteriorRing
geo:geometryType
geo:envelope
geo:getX
geo:getY
geo:equalsExact
geo:isRing
geo:interiorPoint
geo:polygonize
geo:startPoint
geo:endPoint
geo:relatePattern
geo:pointN
geo:equalsExactTolerance
geo:interiorRingN
geo:splitPolygon
geo:length
geo:isEmpty
geo:contains
geo:disjoint
geo:intersects
```

Executing a new Process

Create a Process using a Groovy Closure

```
Process process = new Process("convexhull",
        "Create a convexhull around the features",
        [features: geoscript.layer.Cursor],
        [result: geoscript.layer.Cursor],
        { inputs ->
            def geoms = new GeometryCollection(inputs.features.collect{f -> f.geom})
            def output = new Layer()
            output.add([geoms.convexHull])
            [result: output]
        }
)
String name = process.name
println name
```

```
geoscript:convexhull
```

Get the title

```
String title = process.title
println title
```

```
convexhull
```

Get the description

```
String description = process.description
println description
```

```
Create a convexhull around the features
```

Get the version

```
String version = process.version
println version
```

```
1.0.0
```

Get the input parameters

```
Map parameters = process.parameters
println parameters
```

```
[features:class geoscript.layer.Cursor]
```

Get the output parameters

```
Map results = process.results
println results
```

```
[result:class geoscript.layer.Cursor]
```

Execute the Process created from a Groovy Closure

```
Workspace workspace = new GeoPackage('src/main/resources/data.gpkg')
Layer layer = workspace.get("places")
Map executeResults = process.execute([features: layer.cursor])
Cursor convexHullCursor = executeResults.result
```