Geoscript Groovy Cookbook

Jared Erickson

Table of Contents

Introduction

The GeoScript Groovy Cookbook contains short recipes on how to use the GeoScript Groovy library.

GeoScript is a geospatial library written in Groovy. It provides modules for working with geometries, projections, features, layers, rasters, styles, rendering, and tiles. It is built on top of the Java Topology Suite (JTS) and GeoTools libraries. GeoScript Groovy is open source and licensed under the MIT license.

Geometry Recipes

The Geometry classes are in the **geoscript.geom** package.

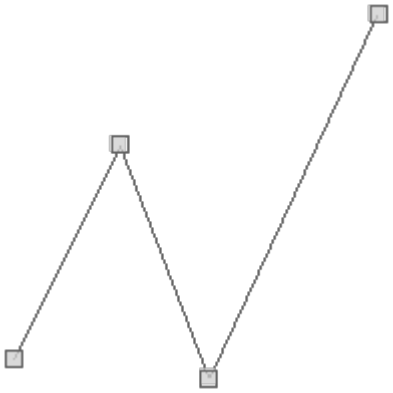
Creating Geometries

Create a Point with an XY

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

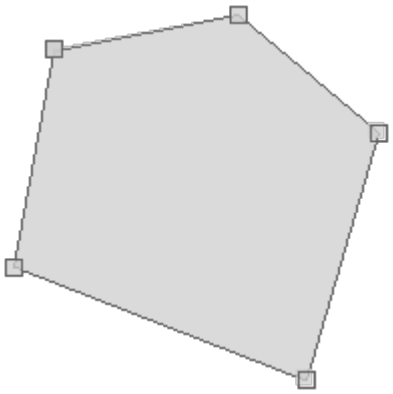
Create a LineString from Coordinates

```
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]
]])
```

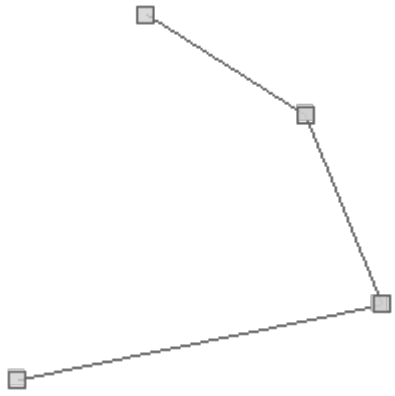


Create a MultiPoint with a List of Points

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

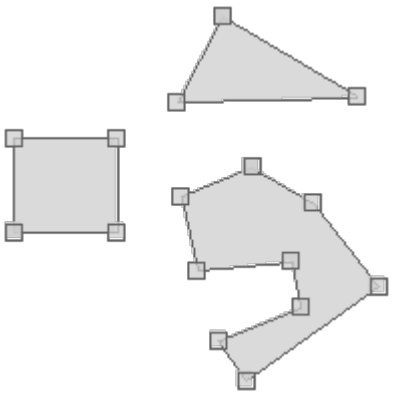
Create a MultiLineString with a List of LineStrings

```
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]
        )
])
```



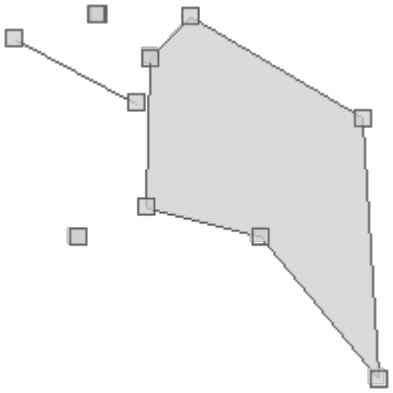
Create a MultiPolygon with a List of Polygons

```
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, 47.47823216312885],
            [-122.20367431640624, 47.543163654317304]
    ]])
)
```



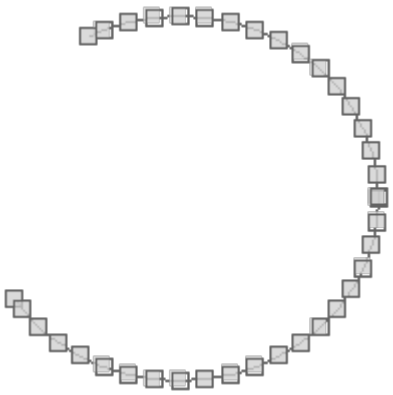
Create a GeometryCollection with a List of Geometries

```
GeometryCollection geometryCollection = new GeometryCollection(
    new LineString ([-157.044, 58.722], [-156.461, 58.676]),
    new Point(-156.648, 58.739),
    new Polygon([[
         [-156.395, 58.7083],
         [-156.412, 58.6026],
         [-155.874, 58.5825],
         [-155.313, 58.4822],
         [-155.385, 58.6655],
         [-156.203, 58.7368],
         [-156.395, 58.7083]
    ]]),
    new Point(-156.741, 58.582)
)
```



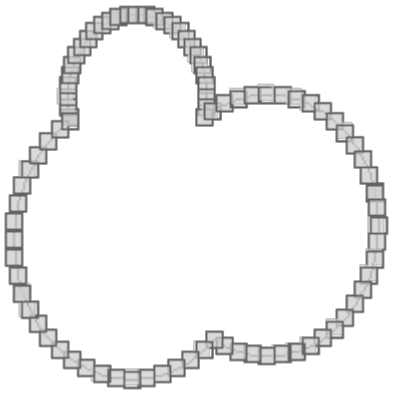
Create a CircularString with a List of Points

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



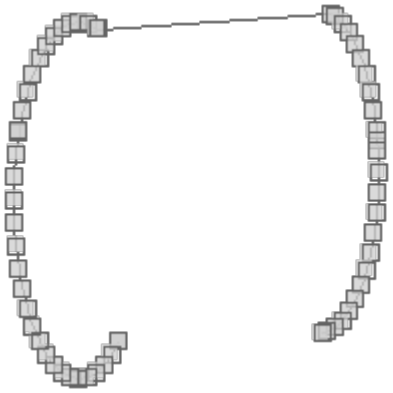
Create a CircularRing with a List of Points

```
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]
])
```



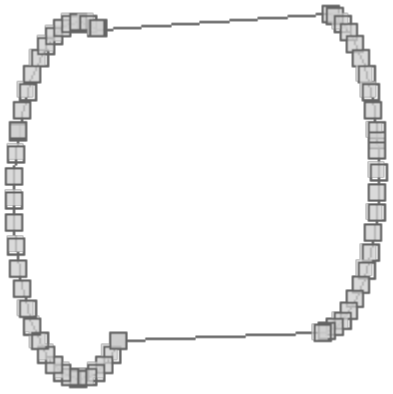
Create a CompoundCurve with a List of CircularStrings and LineStrings

```
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
```

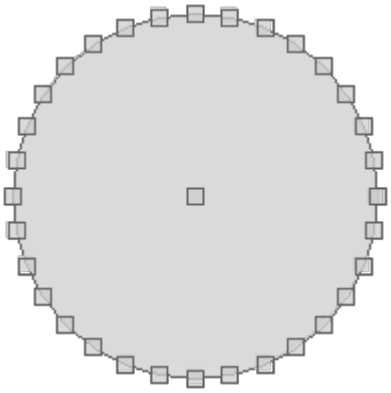
Get the length of a Geometry

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

```
23.24738479915536
```

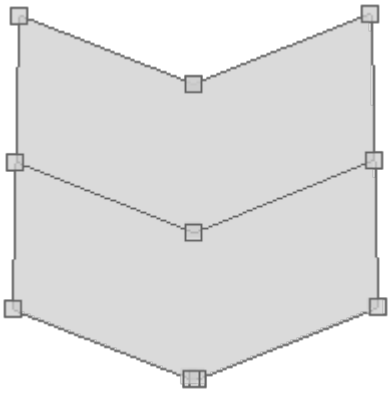
Buffer a Point

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



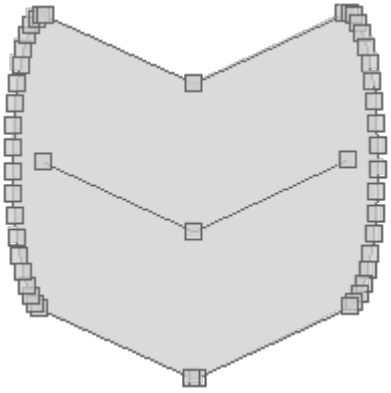
Buffer a LineString with a butt cap

```
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)
```



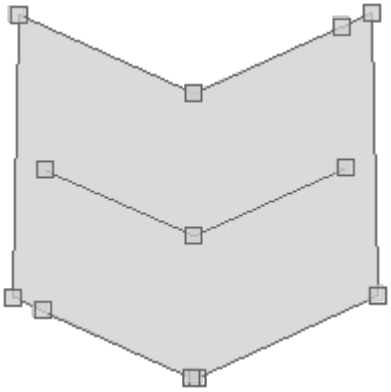
Buffer a LineString with a round cap

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



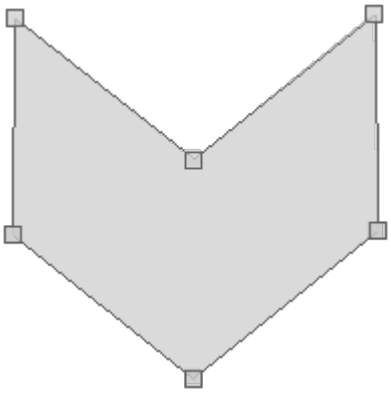
Buffer a LineString with a square cap

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



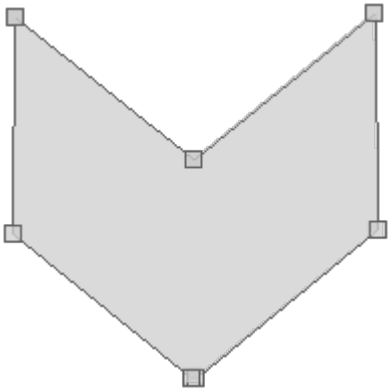
Buffer a LineString on the right side only

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



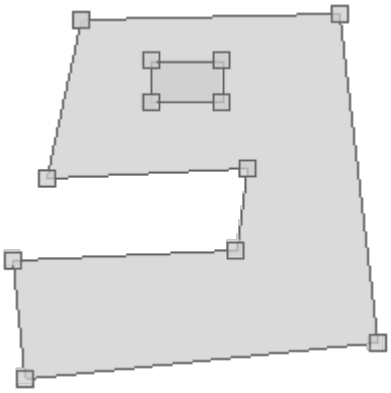
Buffer a LineString on the left side only

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



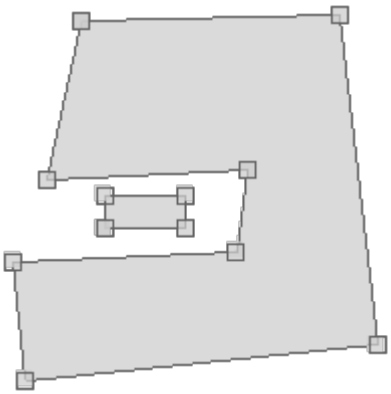
Check whether a Geometry contains another Geometry

```
Polygon polygon1 = new Polygon([[
    [-120.739, 48.151],
    [-121.003, 47.070],
    [-119.465, 47.137],
    [-119.553, 46.581],
    [-121.267, 46.513],
    [-121.168, 45.706],
    [-118.476, 45.951],
    [-118.762, 48.195],
    [-120.739, 48.151]
]])

Polygon polygon2 = new Polygon([[
    [-120.212, 47.591],
    [-119.663, 47.591],
    [-119.663, 47.872],
    [-120.212, 47.872],
    [-120.212, 47.591]
]])

boolean contains = polygon1.contains(polygon2)
println contains
```



```
true
```

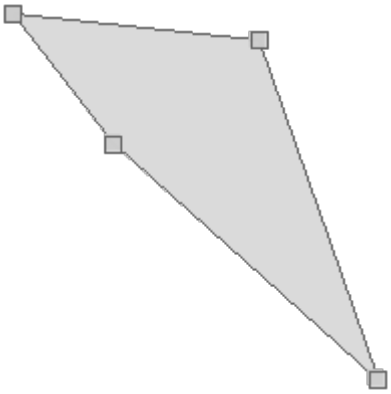
```
Polygon polygon3 = new Polygon([[
    [-120.563, 46.739],
    [-119.948, 46.739],
    [-119.948, 46.965],
    [-120.563, 46.965],
    [-120.563, 46.739]
]])

contains = polygon1.contains(polygon3)
println contains
```



```
false
```

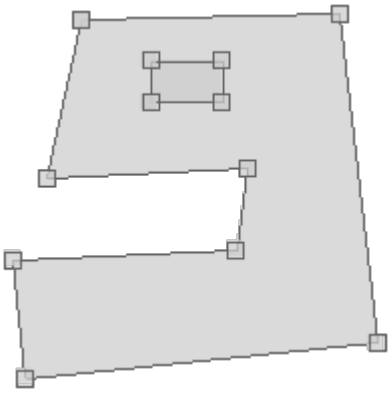
Create a convexhull Geometry around a Geometry

```
Geometry geometry = new MultiPoint(
    new Point(-119.882, 47.279),
    new Point(-100.195, 46.316),
    new Point(-111.796, 42.553),
    new Point(-90.7031, 34.016)
)
Geometry convexHull = geometry.convexHull
```



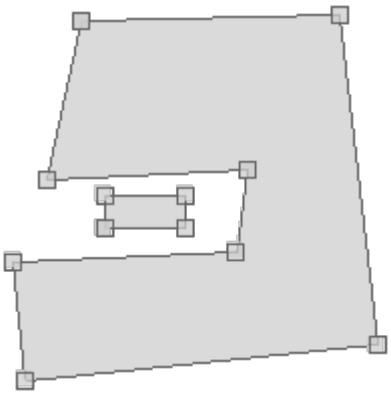
Check whether a Geometry covers another Geometry

```
Polygon polygon1 = new Polygon([[
        [-120.739, 48.151],
        [-121.003, 47.070],
        [-119.465, 47.137],
        [-119.553, 46.581],
        [-121.267, 46.513],
        [-121.168, 45.706],
        [-118.476, 45.951],
        [-118.762, 48.195],
        [-120.739, 48.151]
]])

Polygon polygon2 = new Polygon([[
        [-120.212, 47.591],
        [-119.663, 47.591],
        [-119.663, 47.872],
        [-120.212, 47.872],
        [-120.212, 47.591]
]])

boolean isCovered = polygon1.covers(polygon2)
println isCovered
```



```
true
```

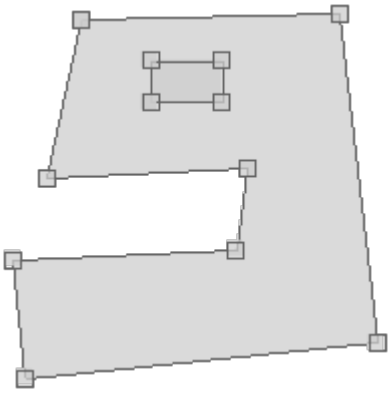
```
Polygon polygon3 = new Polygon([[
        [-120.563, 46.739],
        [-119.948, 46.739],
        [-119.948, 46.965],
        [-120.563, 46.965],
        [-120.563, 46.739]
]])

isCovered = polygon1.covers(polygon3)
println isCovered
```



```
false
```

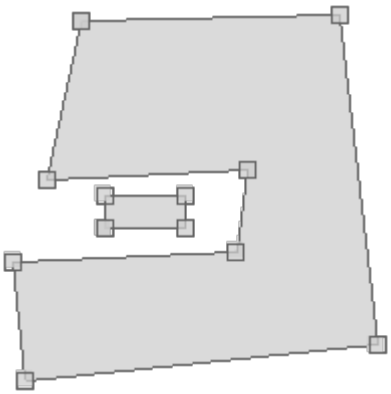
Check whether a Geometry is covered by another Geometry

```
Polygon polygon1 = new Polygon([[
        [-120.739, 48.151],
        [-121.003, 47.070],
        [-119.465, 47.137],
        [-119.553, 46.581],
        [-121.267, 46.513],
        [-121.168, 45.706],
        [-118.476, 45.951],
        [-118.762, 48.195],
        [-120.739, 48.151]
]])

Polygon polygon2 = new Polygon([[
        [-120.212, 47.591],
        [-119.663, 47.591],
        [-119.663, 47.872],
        [-120.212, 47.872],
        [-120.212, 47.591]
]])

boolean isCoveredBy = polygon2.coveredBy(polygon1)
println isCoveredBy
```



```
true
```

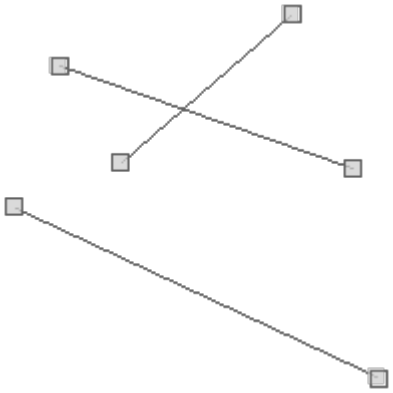
```
Polygon polygon3 = new Polygon([[
        [-120.563, 46.739],
        [-119.948, 46.739],
        [-119.948, 46.965],
        [-120.563, 46.965],
        [-120.563, 46.739]
]])

isCoveredBy = polygon3.coveredBy(polygon1)
println isCoveredBy
```



```
false
```

Check whether one Geometry crosses another Geometry

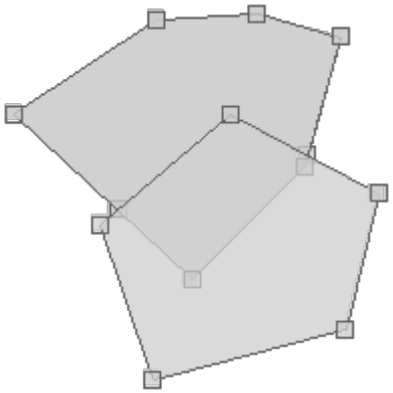
```
LineString line1 = new LineString([[-122.486, 47.256], [-121.695, 46.822]])
LineString line2 = new LineString([[-122.387, 47.613], [-121.750, 47.353]])
LineString line3 = new LineString([[-122.255, 47.368], [-121.882, 47.746]])

boolean doesCross12 = line1.crosses(line2)
println doesCross12

boolean doesCross13 = line1.crosses(line3)
println doesCross13

boolean doesCross23 = line2.crosses(line3)
println doesCross23
```



```
false
false
true
```

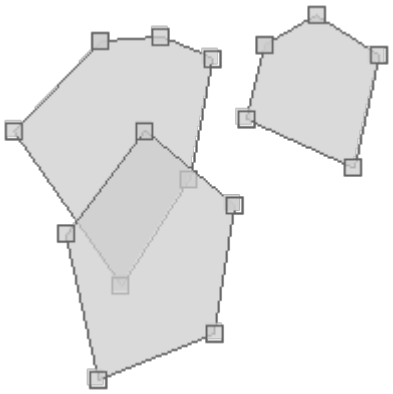
Calculate the difference between two Geometries

```
Polygon polygon1 = new Polygon([[
        [-121.915, 47.390],
        [-122.640, 46.995],
        [-121.739, 46.308],
        [-121.168, 46.777],
        [-120.981, 47.316],
        [-121.409, 47.413],
        [-121.915, 47.390]
]])

Polygon polygon2 = new Polygon([[
        [-120.794, 46.664],
        [-121.541, 46.995],
        [-122.200, 46.536],
        [-121.937, 45.890],
        [-120.959, 46.096],
        [-120.794, 46.664]
]])

Geometry difference = polygon1.difference(polygon2)
```



Check whether one Geometry is disjoint from another Geometry

```
Polygon polygon1 = new Polygon([[
        [-121.915, 47.390],
        [-122.640, 46.995],
        [-121.739, 46.308],
        [-121.168, 46.777],
        [-120.981, 47.316],
        [-121.409, 47.413],
        [-121.915, 47.390]
]])

Polygon polygon2 = new Polygon([[
        [-120.794, 46.664],
        [-121.541, 46.995],
        [-122.200, 46.536],
        [-121.937, 45.890],
        [-120.959, 46.096],
        [-120.794, 46.664]
]])

Polygon polygon3 = new Polygon([[
        [-120.541, 47.376],
        [-120.695, 47.047],
        [-119.794, 46.830],
        [-119.586, 47.331],
        [-120.102, 47.509],
        [-120.541, 47.376]
]])

boolean isDisjoint12 = polygon1.disjoint(polygon2)
println isDisjoint12

boolean isDisjoint13 = polygon1.disjoint(polygon3)
println isDisjoint13

boolean isDisjoint23 = polygon2.disjoint(polygon3)
println isDisjoint23
```



```
false
true
true
```

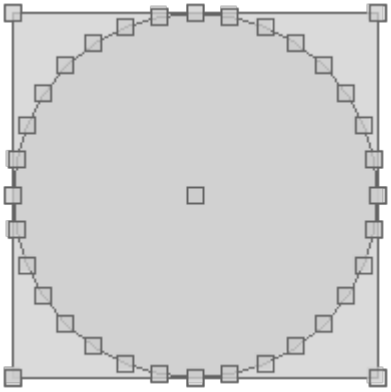
Calculate the distance bewteen two Geometries

```
Point point1 = new Point(-122.442, 47.256)
Point point2 = new Point(-122.321, 47.613)
double distance = point1.distance(point2)
println distance
```

```
0.37694827231332195
```

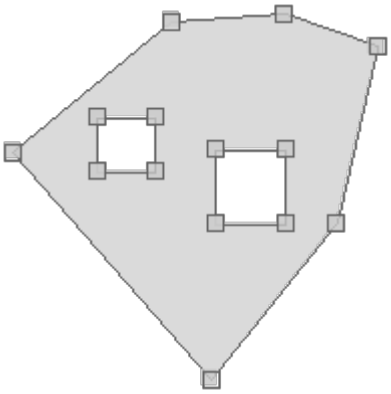
Get Bounds from a Geometry

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



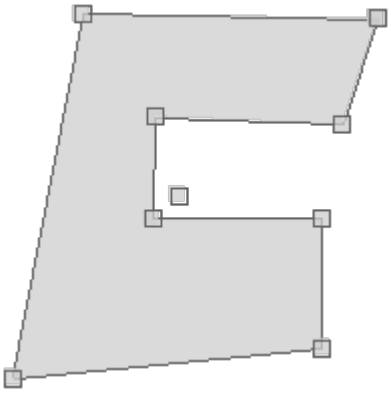
Get the Boundary from a Geometry

```
Polygon polygon = new Polygon([
    [
            [-121.915, 47.390],
            [-122.640, 46.995],
            [-121.739, 46.308],
            [-121.168, 46.777],
            [-120.981, 47.316],
            [-121.409, 47.413],
            [-121.915, 47.390]
    ],
    [
            [-122.255, 46.935],
            [-121.992, 46.935],
            [-121.992, 47.100],
            [-122.255, 47.100],
            [-122.255, 46.935]
    ],
    [
            [-121.717, 46.777],
            [-121.398, 46.777],
            [-121.398, 47.002],
            [-121.717, 47.002],
            [-121.717, 46.777]
    ]
])
Geometry boundary = polygon.boundary
```



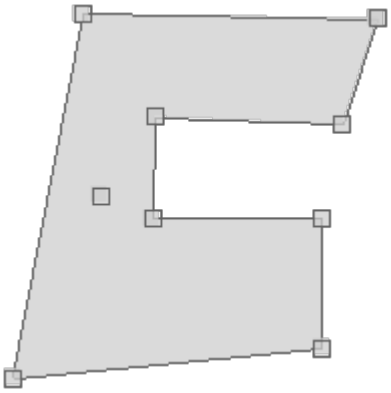
Get the Centroid from a Geometry

```
Polygon polygon = new Polygon([[
       [-118.937, 48.327],
       [-121.157, 48.356],
       [-121.684, 46.331],
       [-119.355, 46.498],
       [-119.355, 47.219],
       [-120.629, 47.219],
       [-120.607, 47.783],
       [-119.201, 47.739],
       [-118.937, 48.327]
]])
Geometry centroid = polygon.centroid
```



Get the Interior Point from a Geometry

```
Polygon polygon = new Polygon([[
       [-118.937, 48.327],
       [-121.157, 48.356],
       [-121.684, 46.331],
       [-119.355, 46.498],
       [-119.355, 47.219],
       [-120.629, 47.219],
       [-120.607, 47.783],
       [-119.201, 47.739],
       [-118.937, 48.327]
]])
Geometry interiorPoint = polygon.interiorPoint
```



Get the number of Geometries

```
MultiPoint multiPoint = new MultiPoint([
        new Point(-122.3876953125, 47.5820839916191),
        new Point(-122.464599609375, 47.25686404408872),
        new Point(-122.48382568359374, 47.431803338643334)
])
int number = multiPoint.numGeometries
println number
```

```
3
```

Get a Geometry by index

```
MultiPoint multiPoint = new MultiPoint([
        new Point(-122.3876953125, 47.5820839916191),
        new Point(-122.464599609375, 47.25686404408872),
        new Point(-122.48382568359374, 47.431803338643334)
])
(0..<multiPoint.getNumGeometries()).each { int i ->
    Geometry geometry = multiPoint.getGeometryN(i)
    println geometry.wkt
}
```

```
POINT (-122.3876953125 47.5820839916191)
POINT (-122.464599609375 47.25686404408872)
POINT (-122.48382568359374 47.431803338643334)
```

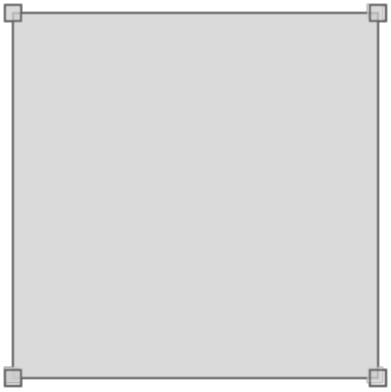
Get a List of Geometries

```
MultiPoint multiPoint = new MultiPoint([
        new Point(-122.3876953125, 47.5820839916191),
        new Point(-122.464599609375, 47.25686404408872),
        new Point(-122.48382568359374, 47.431803338643334)
])
List<Geometry> geometries = multiPoint.geometries
geometries.each { Geometry geometry ->
    println geometry.wkt
}
```

```
POINT (-122.3876953125 47.5820839916191)
POINT (-122.464599609375 47.25686404408872)
POINT (-122.48382568359374 47.431803338643334)
```

Get the number of Points in a Geometry

```
Polygon polygon = new Polygon([[
    [-120.563, 46.739],
    [-119.948, 46.739],
    [-119.948, 46.965],
    [-120.563, 46.965],
    [-120.563, 46.739]
]])
int number = polygon.numPoints
println number
```



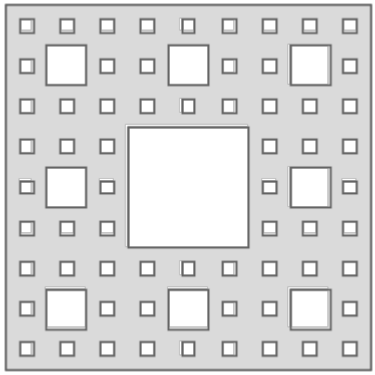
```
5
```

Create a Geometry of a String

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

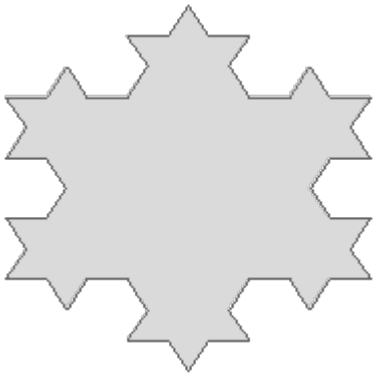
Create a Sierpinski Carpet in a given Bounds and with a number of points

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



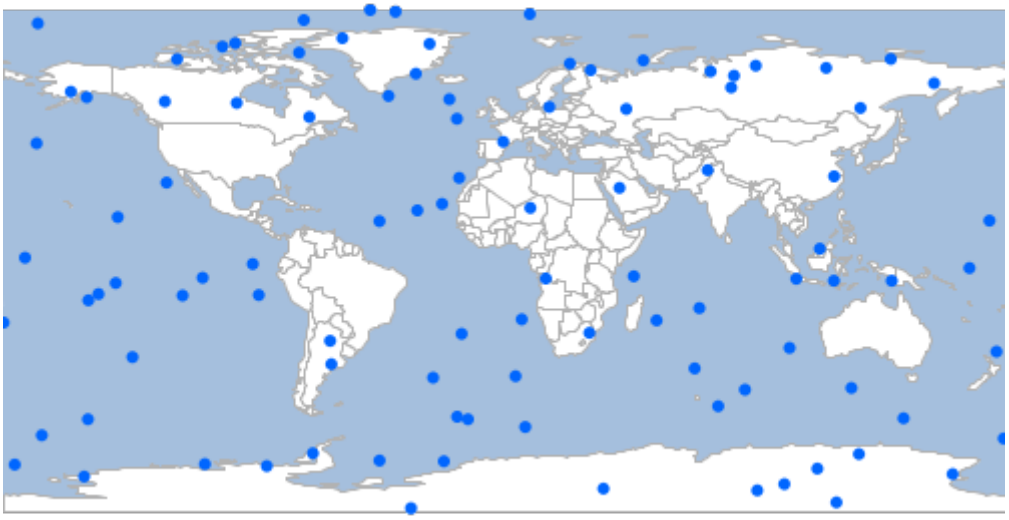
Create a Kock Snowflake in a given Bounds and with a number of points

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



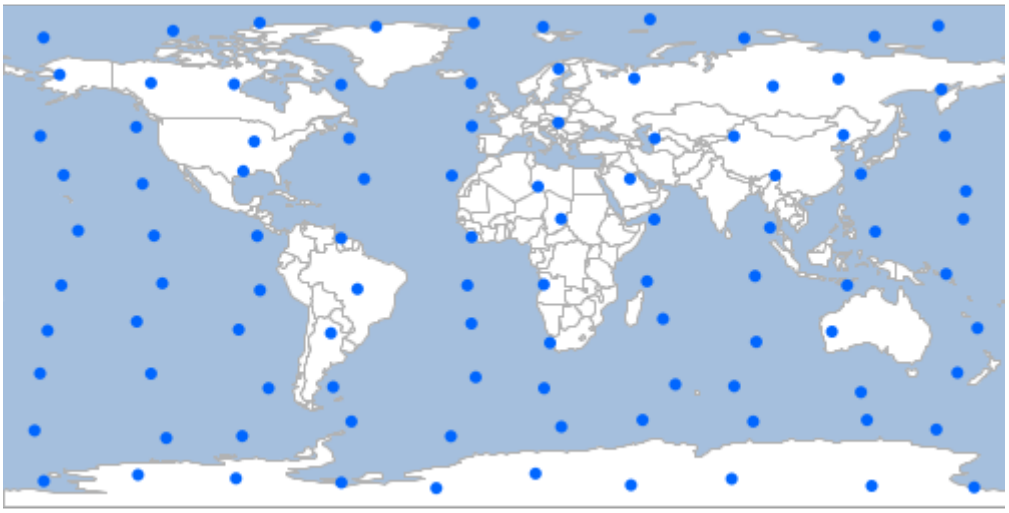
Create a number of random points within a given Geometry

```
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]}
```

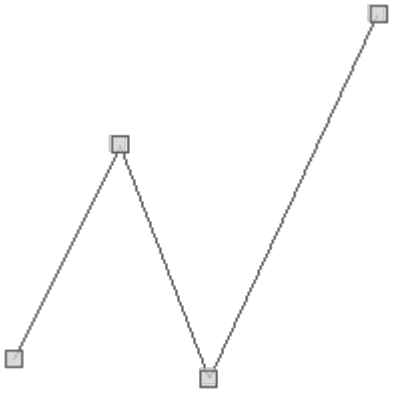
WKB

Read a Geometry from WKB using the WkbReader

```
String wkb = "0000000001C05EC9999999999A40471E5604189375"
WkbReader reader = new WkbReader()
Geometry geometry = reader.read(wkb)
```

Read a Geometry from WKB using the Geometry.fromWKB() static method

```
String wkb =
"00000000020000000440099581062 4DD2F404594FDF3B645A2401ADA1CAC0831274048E0A3D70A3D71402
3676C8B43958140454BC6A7EF9DB2402EA3D70A3D70A4404AE624DD2F1AA0"
Geometry geometry = Geometry.fromWKB(wkb)
```



Get the WKB of a Geometry

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

```
0000000001C05EC9999999999A40471E5604189375
```

Write a Geometry to WKB using the WkbWriter

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

```
00000000020000000440099581062 4DD2F404594FDF3B645A2401ADA1CAC0831274048E0A3D70A3D714023
676C8B43958140454BC6A7EF9DB2402EA3D70A3D70A4404AE624DD2F1AA0
```

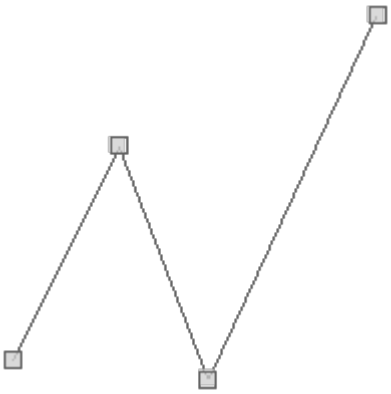
WKT

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)
```

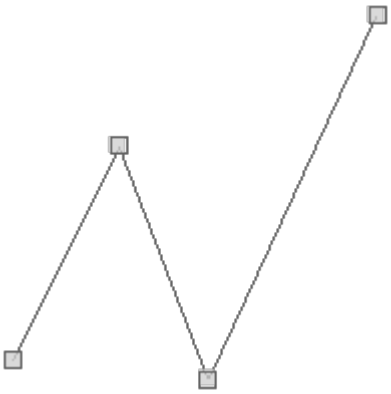
GeoJSON

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]]}
```

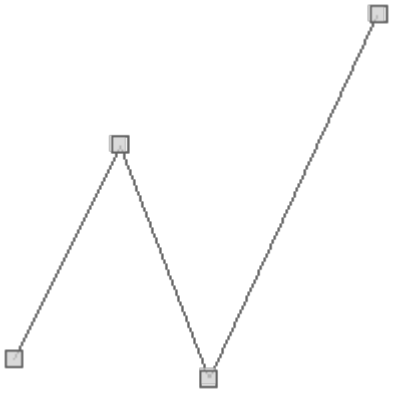
KML

Read a Geometry from KML using the KmlReader

```
String kml = "<Point><coordinates>-123.15,46.237</coordinates></Point>"
KmlReader reader = new KmlReader()
Geometry geometry = reader.read(kml)
```

Read a Geometry from KML using the Geometry.fromKml() static method

```
String kml = "<LineString><coordinates>3.198,43.164 6.713,49.755 9.702,42.592
15.32,53.798</coordinates></LineString>"
Geometry geometry = Geometry.fromKml(kml)
```



Get the KML of a Geometry

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

```
<Point><coordinates>-123.15,46.237</coordinates></Point>
```

Write a Geometry to KML using the KmlWriter

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

```
<LineString><coordinates>3.198,43.164 6.713,49.755 9.702,42.592
15.32,53.798</coordinates></LineString>
```

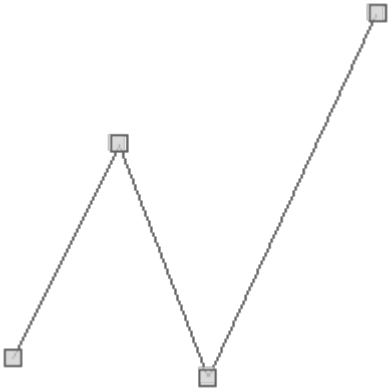
Geobuf

Read a Geometry from Geobuf using the GeobufReader

```
String geobuf = "10021806320c08001a08dffab87590958c2c"
GeobufReader reader = new GeobufReader()
Geometry geometry = reader.read(geobuf)
```

Read a Geometry from Geobuf using the Geometry.fromGeobuf() static method

```
String geobuf =
"10021806322408021a20e0b08603c0859529f089ad03b0c8a40690efec02efb1ea06a0e5ad05e0f5d70a"
Geometry geometry = Geometry.fromGeobuf(geobuf)
```



Get the Geobuf of a Geometry

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

```
10021806320c08001a08dffab87590958c2c
```

Write a Geometry to Geobuf using the GeobufWriter

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

```
10021806322408021a20e0b08603c0859529f089ad03b0c8a40690efec02efb1ea06a0e5ad05e0f5d70a
```

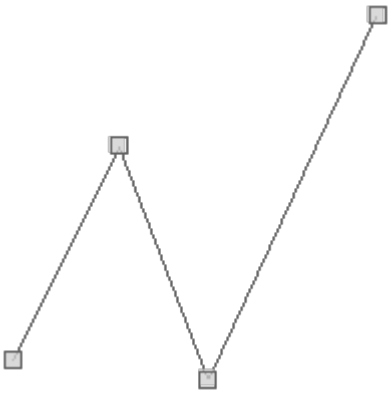
GML 2

Read a Geometry from GML2 using the Gml2Reader

```
String gml2 = "<gml:Point><gml:coordinates>-
123.15,46.237</gml:coordinates></gml:Point>"
Gml2Reader reader = new Gml2Reader()
Geometry geometry = reader.read(gml2)
```

Read a Geometry from GML2 using the Geometry.fromGML2() static method

```
String gml2 = "<gml:LineString><gml:coordinates>3.198,43.164 6.713,49.755 9.702,42.592
15.32,53.798</gml:coordinates></gml:LineString>"
Geometry geometry = Geometry.fromGML2(gml2)
```



Get the GML2 of a Geometry

```
Geometry geometry = new Point(-123.15, 46.237)
String gml2 = geometry.gml2
println gml2
```

```
<gml:Point><gml:coordinates>-123.15,46.237</gml:coordinates></gml:Point>
```

Write a Geometry to GML2 using the Gml2Writer

```
Geometry geometry = new LineString(
        [3.198, 43.164],
        [6.713, 49.755],
        [9.702, 42.592],
        [15.32, 53.798]
)
Gml2Writer writer = new Gml2Writer()
String gml2 = writer.write(geometry)
println gml2
```

```
<gml:LineString><gml:coordinates>3.198,43.164 6.713,49.755 9.702,42.592
15.32,53.798</gml:coordinates></gml:LineString>
```

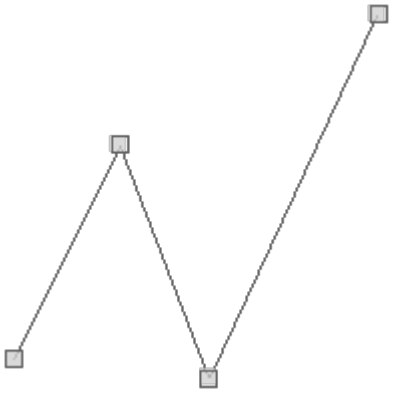
GML 3

Read a Geometry from GML3 using the Gml3Reader

```
String gml3 = "<gml:Point><gml:pos>-123.15 46.237</gml:pos></gml:Point>"
Gml3Reader reader = new Gml3Reader()
Geometry geometry = reader.read(gml3)
```

Read a Geometry from GML3 using the Geometry.fromGML3() static method

```
String gml3 = "<gml:LineString><gml:posList>3.198 43.164 6.713 49.755 9.702 42.592
15.32 53.798</gml:posList></gml:LineString>"
Geometry geometry = Geometry.fromGML3(gml3)
```



Get the GML3 of a Geometry

```
Geometry geometry = new Point(-123.15, 46.237)
String gml3 = geometry.gml3
println gml3
```

```
<gml:Point><gml:pos>-123.15 46.237</gml:pos></gml:Point>
```

Write a Geometry to GML3 using the Gml3Writer

```
Geometry geometry = new LineString(
        [3.198, 43.164],
        [6.713, 49.755],
        [9.702, 42.592],
        [15.32, 53.798]
)
Gml3Writer writer = new Gml3Writer()
String gml3 = writer.write(geometry)
println gml3
```

```
<gml:LineString><gml:posList>3.198 43.164 6.713 49.755 9.702 42.592 15.32
53.798</gml:posList></gml:LineString>
```

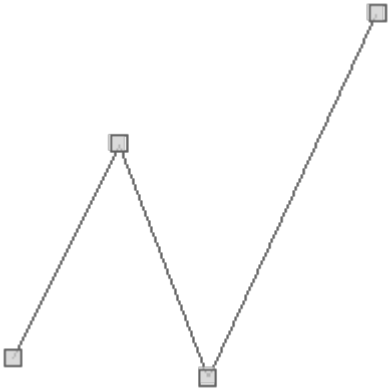
GPX

Read a Geometry from GPX using the GpxReader

```
String gpx = "<wpt lat='46.237' lon='-123.15'/>"
GpxReader reader = new GpxReader()
Geometry geometry = reader.read(gpx)
```

Read a Geometry from GPX using the Geometry.fromGPX() static method

```
String gpx = "<rte><rtept lat='43.164' lon='3.198' /><rtept lat='49.755' lon='6.713'
/><rtept lat='42.592' lon='9.702' /><rtept lat='53.798' lon='15.32' /></rte>"
Geometry geometry = Geometry.fromGpx(gpx)
```



Get the GPX of a Geometry

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

```
<wpt lat='46.237' lon='-123.15'/>
```

Write a Geometry to GPX using the GpxWriter

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

```
<rte><rtept lat='43.164' lon='3.198' /><rtept lat='49.755' lon='6.713' /><rtept
lat='42.592' lon='9.702' /><rtept lat='53.798' lon='15.32' /></rte>
```

GeoRSS

Read a Geometry from GeoRSS using the GeoRSSReader

```
String georss = "<georss:point>46.237 -123.15</georss:point>"
GeoRSSReader reader = new GeoRSSReader()
Geometry geometry = reader.read(georss)
```

Write a Geometry to GeoRSS using the GeoRSSWriter

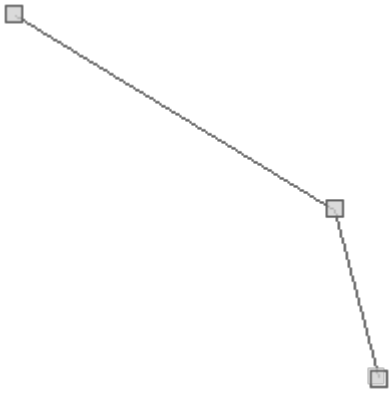
```
Geometry geometry = new LineString(
        [3.198, 43.164],
        [6.713, 49.755],
        [9.702, 42.592],
        [15.32, 53.798]
)
GeoRSSWriter writer = new GeoRSSWriter()
String georss = writer.write(geometry)
println georss
```

```
<georss:line>43.164 3.198 49.755 6.713 42.592 9.702 53.798 15.32</georss:line>
```

Google Polyline

Read a Geometry from a Google Polyline Encoded String using the GeoRSSReader

```
String str = "_p~iF~ps|U_ulLnnqC_mqNvxq`@"
GooglePolylineEncoder encoder = new GooglePolylineEncoder()
Geometry geometry = encoder.read(str)
```



Write a Geometry to a Google Polyline Encoded String using the GeoRSSWriter

```
Geometry geometry = new LineString(
        [3.198, 43.164],
        [6.713, 49.755],
        [9.702, 42.592],
        [15.32, 53.798]
)
GooglePolylineEncoder encoder = new GooglePolylineEncoder()
String str = encoder.write(geometry)
println str
```

```
_nmfGoroRwhfg@womTv_vj@gxfQotkcAogha@
```

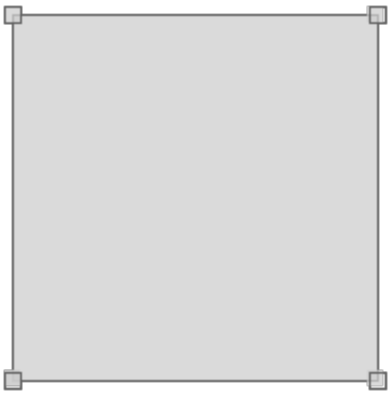
Bounds Recipes

The Bounds class is in the **geoscript.geom** package.

Creating Bounds

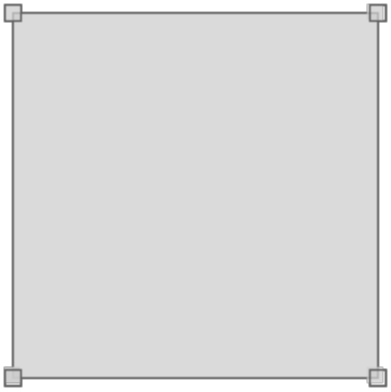
Create a Bounds from four coordinates (minx, miny, maxx, maxy) and a projection.

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



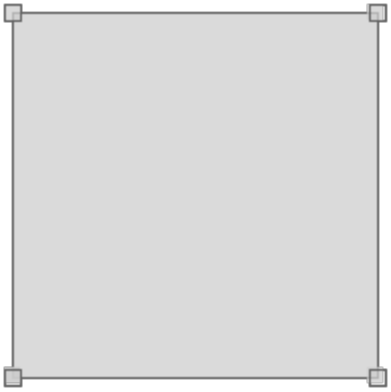
Create a Bounds from four coordinates (minx, miny, maxx, maxy) without a projection. The projection can be set later.

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



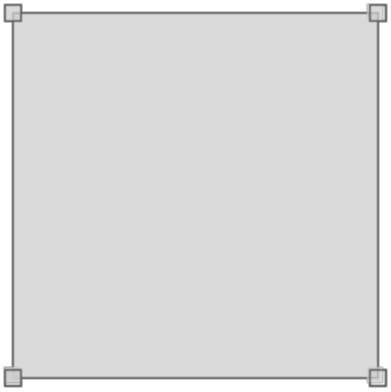
Create a Bounds from a string with commas delimiting minx, miny, maxx, maxy and projection values.

```
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
```

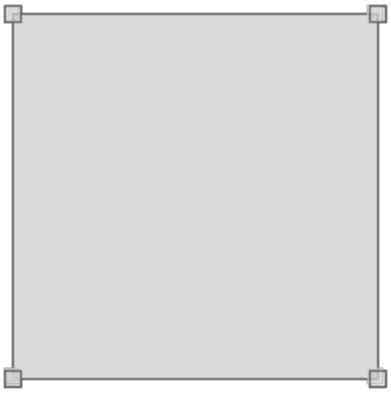
Get the aspect ratio

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

```
1.8987674837280144
```

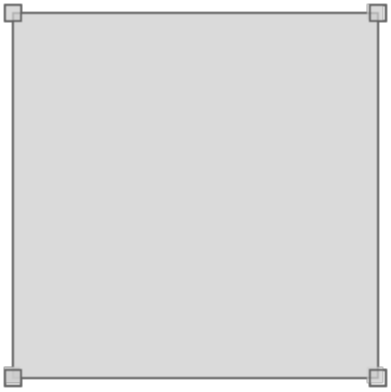
A Bounds is not a Geometry but you can get a Geometry from a Bounds

```
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
```

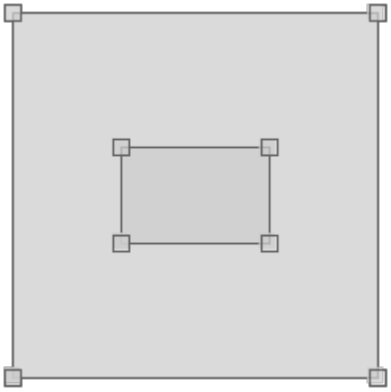
```
(-122.485,47.246,-122.452,47.267,EPSG:4326)
```

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

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

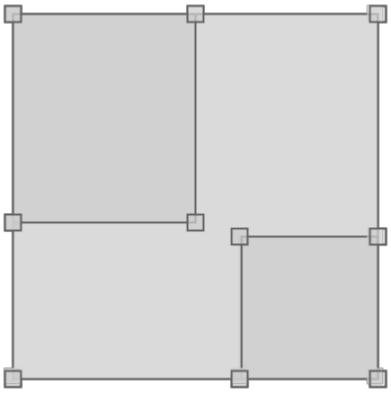
Expand a Bounds by a given distance

```
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)
```



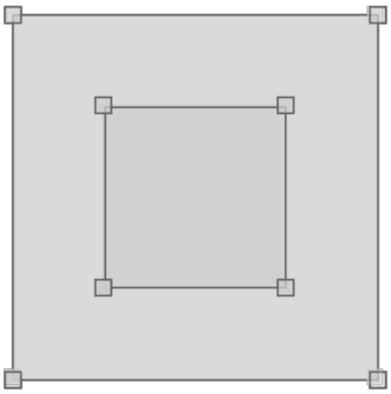
Expand a Bounds to include another Bounds

```
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)
```



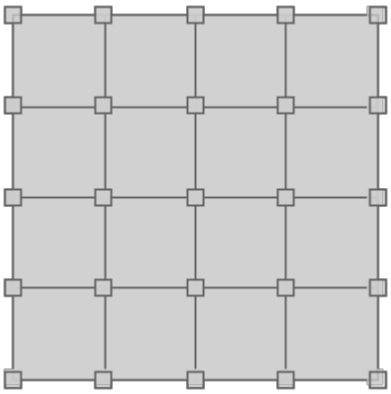
Scale an existing Bounds some distance to create a new Bounds

```
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()
```

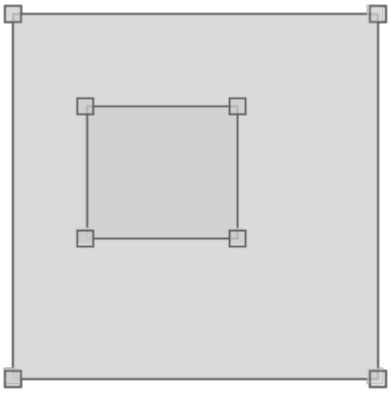
```
false
```

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

```
true
```

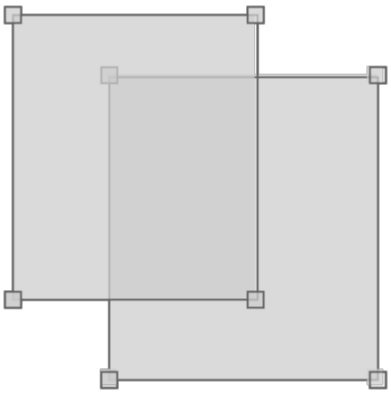
Determine if a Bounds contains another Bounds

```
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
```

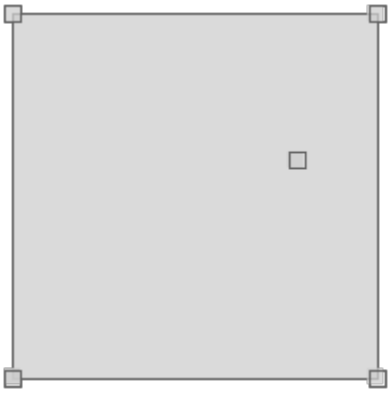
```
Bounds bounds3 = new Bounds(-112.412, 36.809, -99.316, 44.777)
println bounds1.contains(bounds3)
```



```
false
```

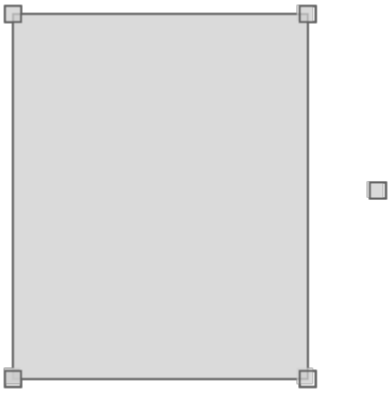
Determine if a Bounds contains a Point

```
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
```

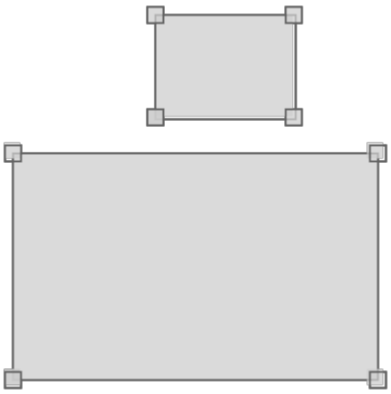
```
Point point2 = new Point(-89.384, 38.959)
println bounds.contains(point2)
```



```
true
```

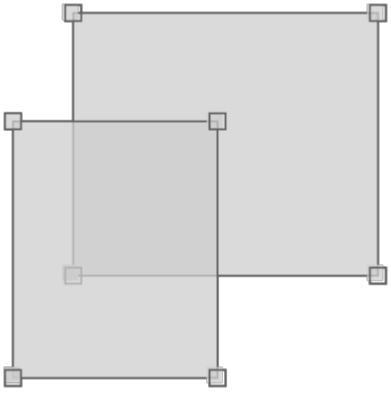
Determine if two Bounds intersect

```
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
```

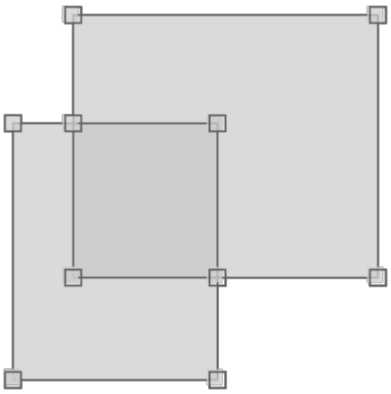
```
Bounds bounds3 = new Bounds(-95.904, 46.747, -95.839, 46.792)
println bounds1.intersects(bounds3)
```



```
true
```

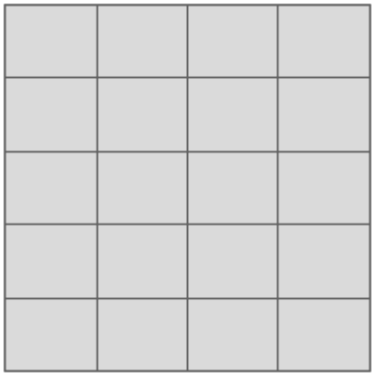
Calculate the intersection between two Bounds

```
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)
```



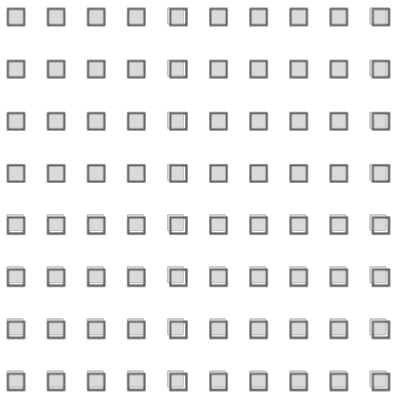
Generate a grid from a Bounds with a given number of columns and rows and the polygon shape. Other shapes include: polygon, point, circle/ellipse, hexagon, hexagon-inv).

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



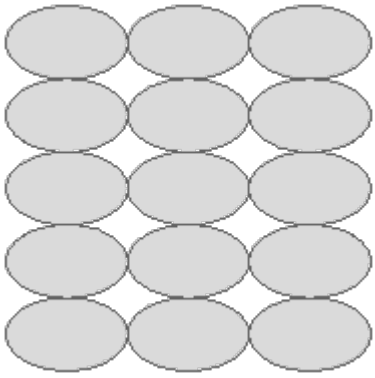
Generate a grid from a Bounds with a given number of columns and rows and a point shape. A Closure that is called with a geometry, column, and row for each grid cell that is created.

```
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)
}
```



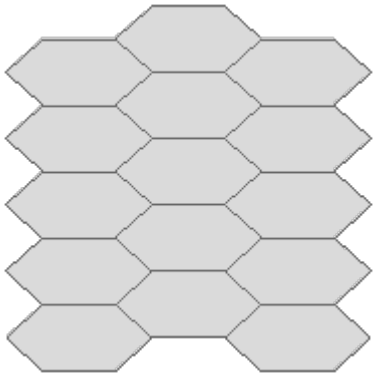
Generate a grid from a Bounds with a given cell width and height and a circle/ellipse shape.

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



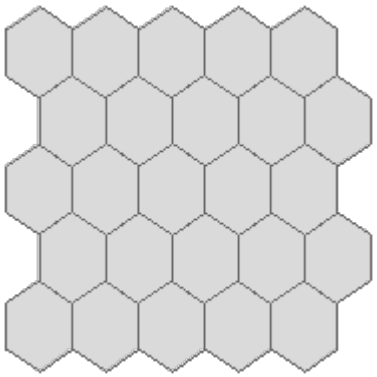
Generate a grid from a Bounds with a given cell width and height and a hexagon shape. A Closure is called with a geometry, column, and row for each grid cell generated.

```
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)
}
```



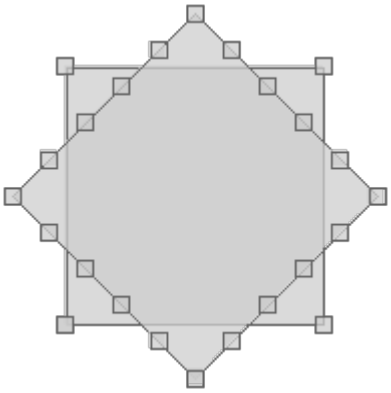
Generate a grid from a Bounds with a given cell width and height and an inverted hexagon shape.

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



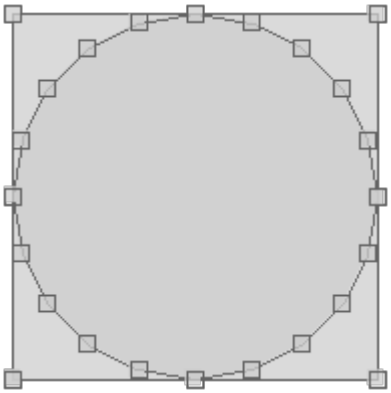
Create a rectangle from a Bounds with a given number of Points and a rotation angle in radians.

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



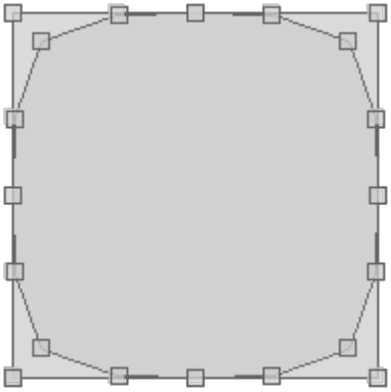
Create an ellipse from a Bounds. The default number of points is 20 and the default rotation angle in radians is 0.

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



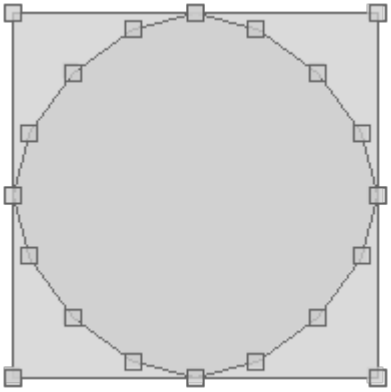
Create a squircle from a Bounds. The default number of points is 20 and the default rotation angle in radians is 0.

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



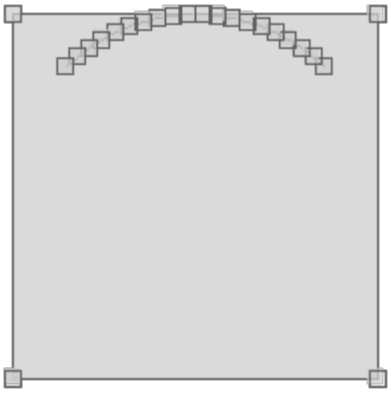
Create a super circle from a Bounds with a given power. The default number of points is 20 and the default rotation angle in radians is 0.

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



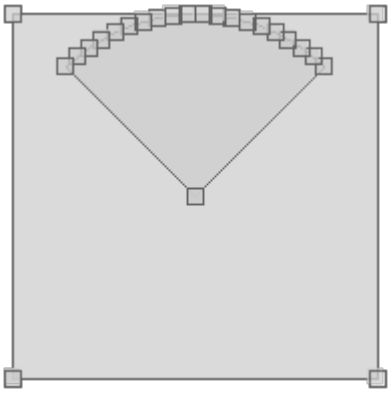
Create an arc from a Bounds with a start angle and angle extent. The default number of points is 20 and the default rotation angle in radians is 0.

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



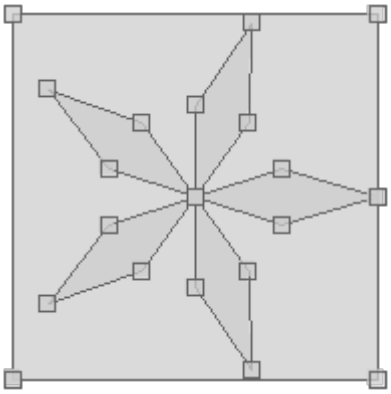
Create an arc polygon from a Bounds with a start angle and angle extent. The default number of points is 20 and the default rotation angle in radians is 0.

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



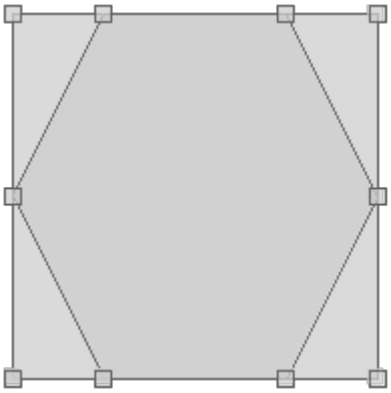
Create a sine star from a Bounds with a number of arms and an arm length ratio. The default number of points is 20 and the default rotation angle in radians is 0.

```
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

The Projection classes are in the **geoscript.proj** package.

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["False easting", 0.0],
  PARAMETER["False northing", 0.0],
  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

The Index classes are in the **geoscript.index** package.

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
```

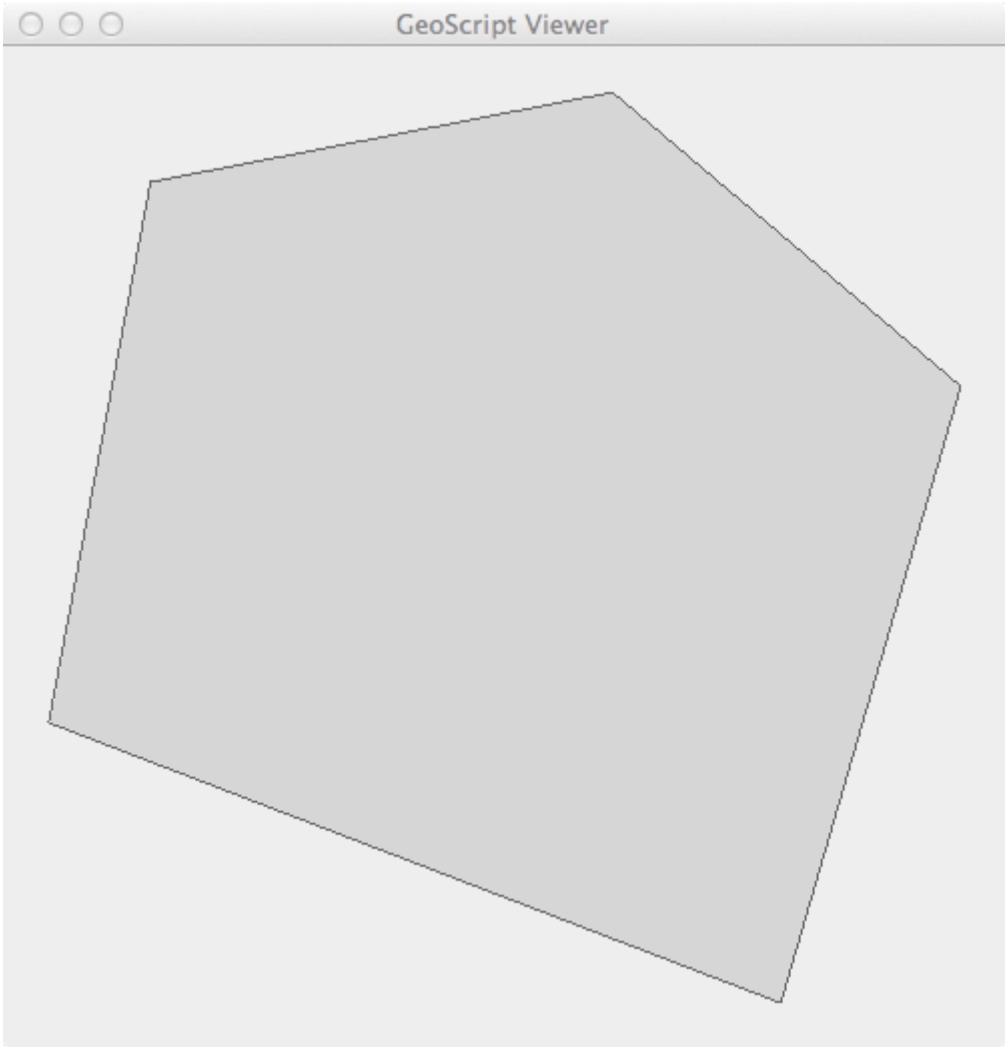
Viewer Recipes

The Viewer classes are in the **geoscript.viewer** package.

Drawing geometries

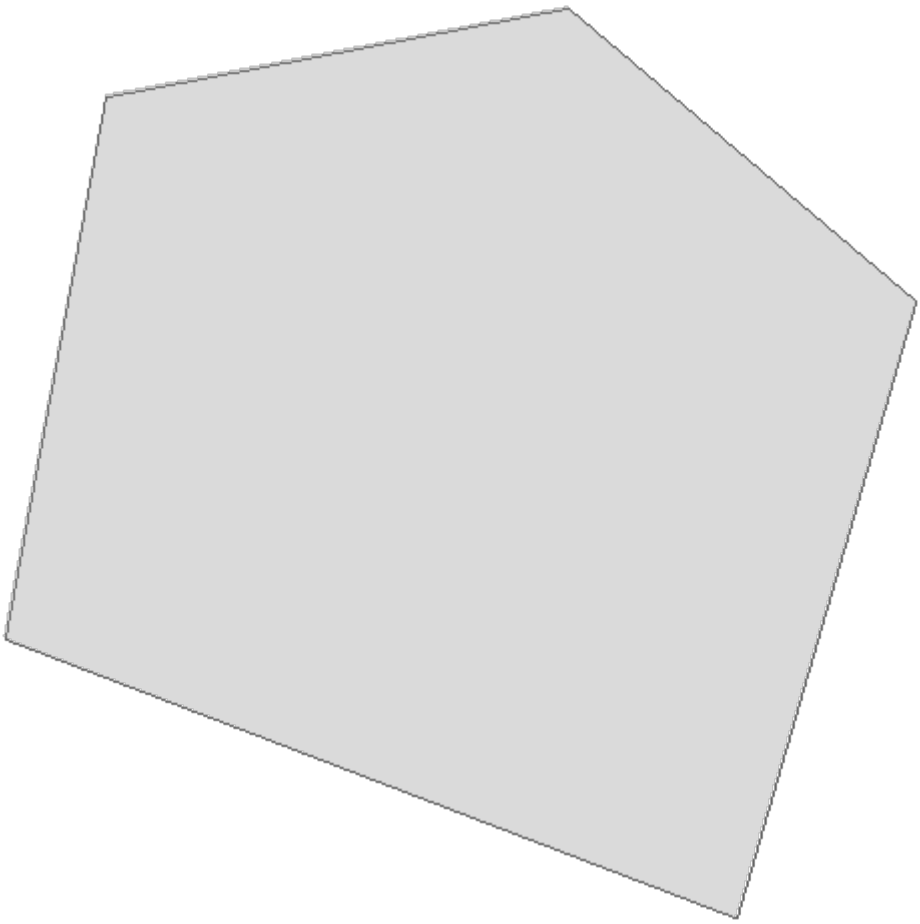
Draw a geometry in a simple GUI

```
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)
```



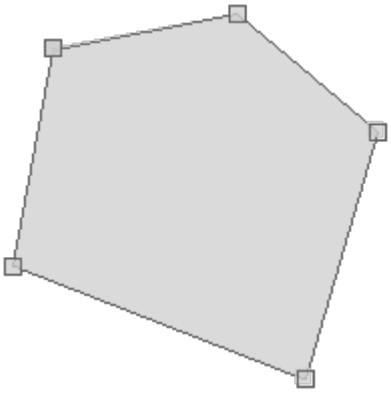
Draw a geometry to an image

```
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)
```



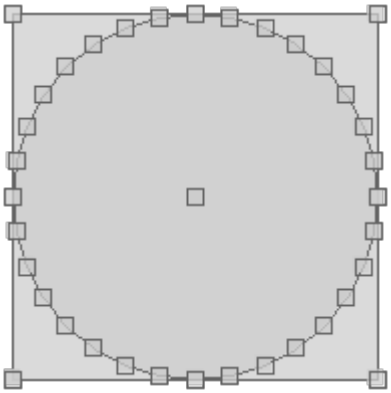
Draw a geometry to an image with options

```
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
)
```



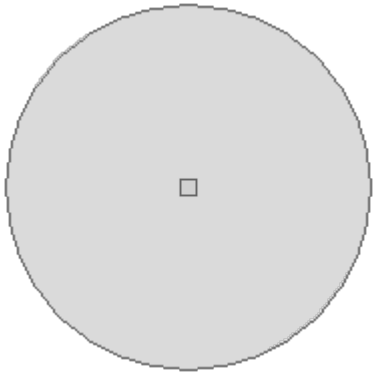
Draw a List of geometries to an image

```
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

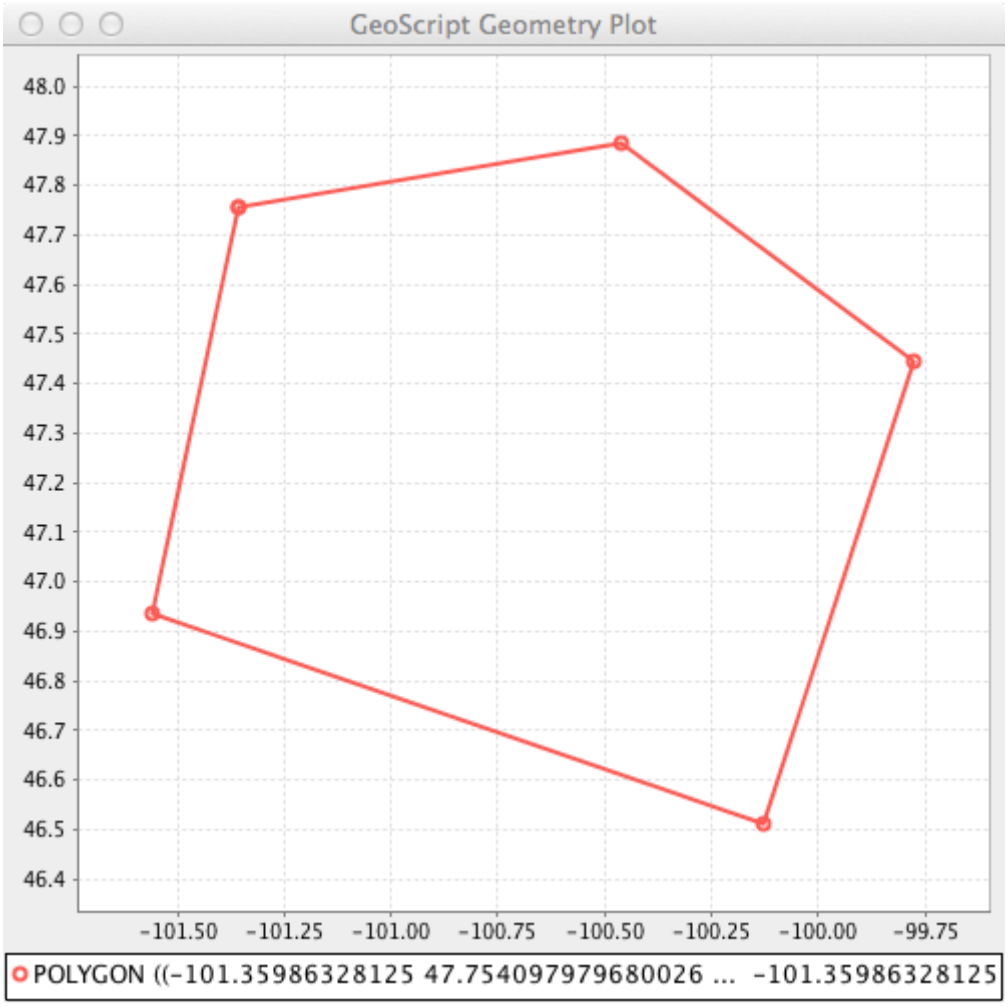
```
Point point = new Point(-123.11, 47.23)
Geometry buffer = point.buffer(4)
File file = new File("geometry.png")
Viewer.drawToFile([buffer, point], file, size: [200,200])
```



Plotting geometries

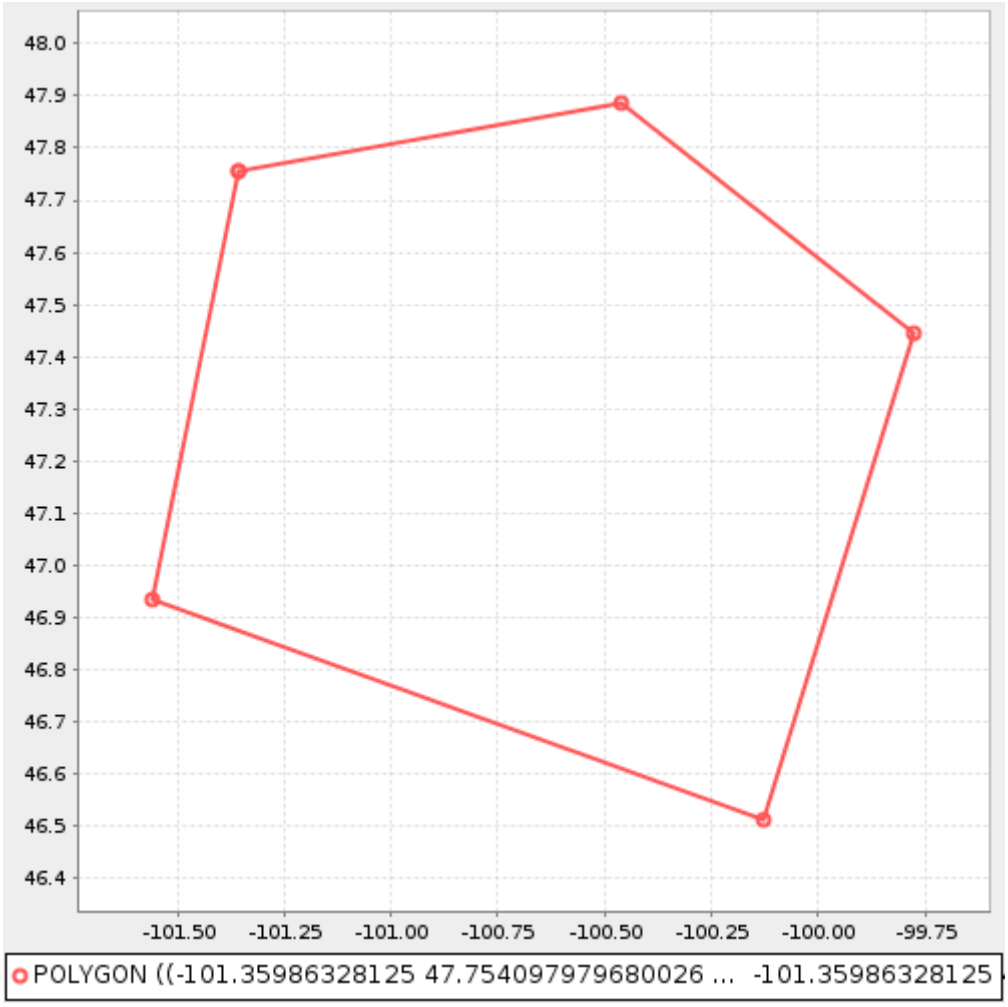
Plot a geometry in a simple GUI

```
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)
```



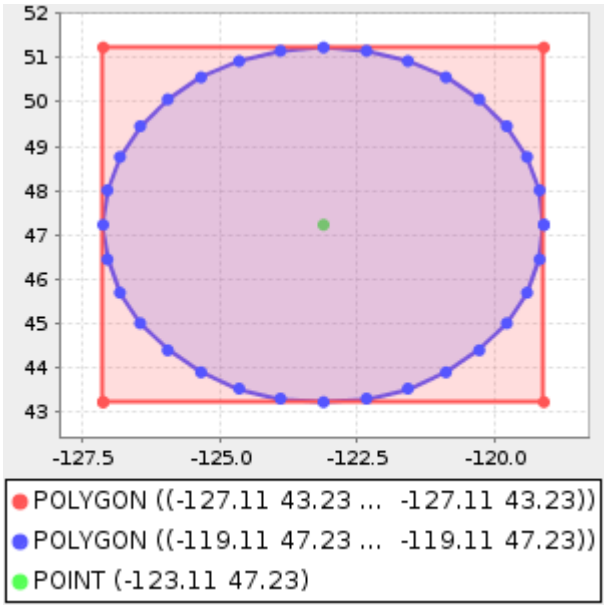
Plot a Geometry to an image

```
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)
```



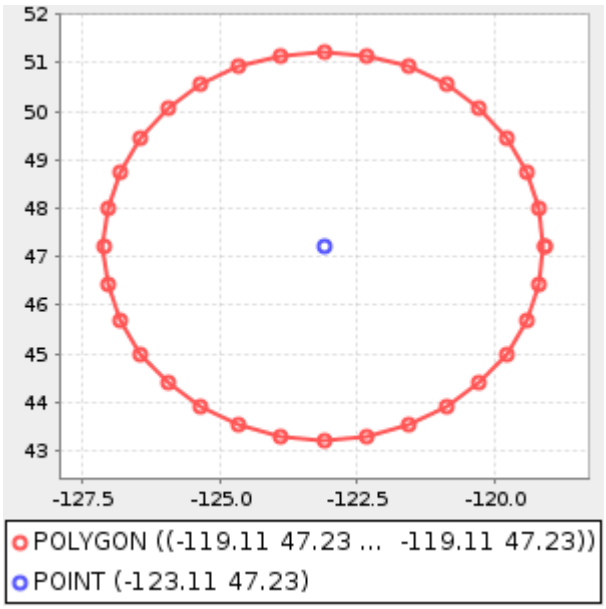
Plot a List of Geometries to an image

```
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])
```



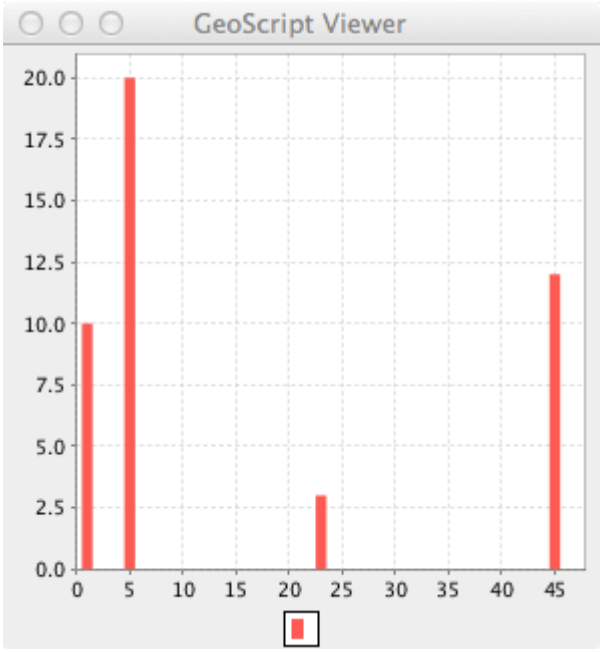
Plot Recipes

The Plot classes are in the **geoscript.plot** package.

Processing Charts

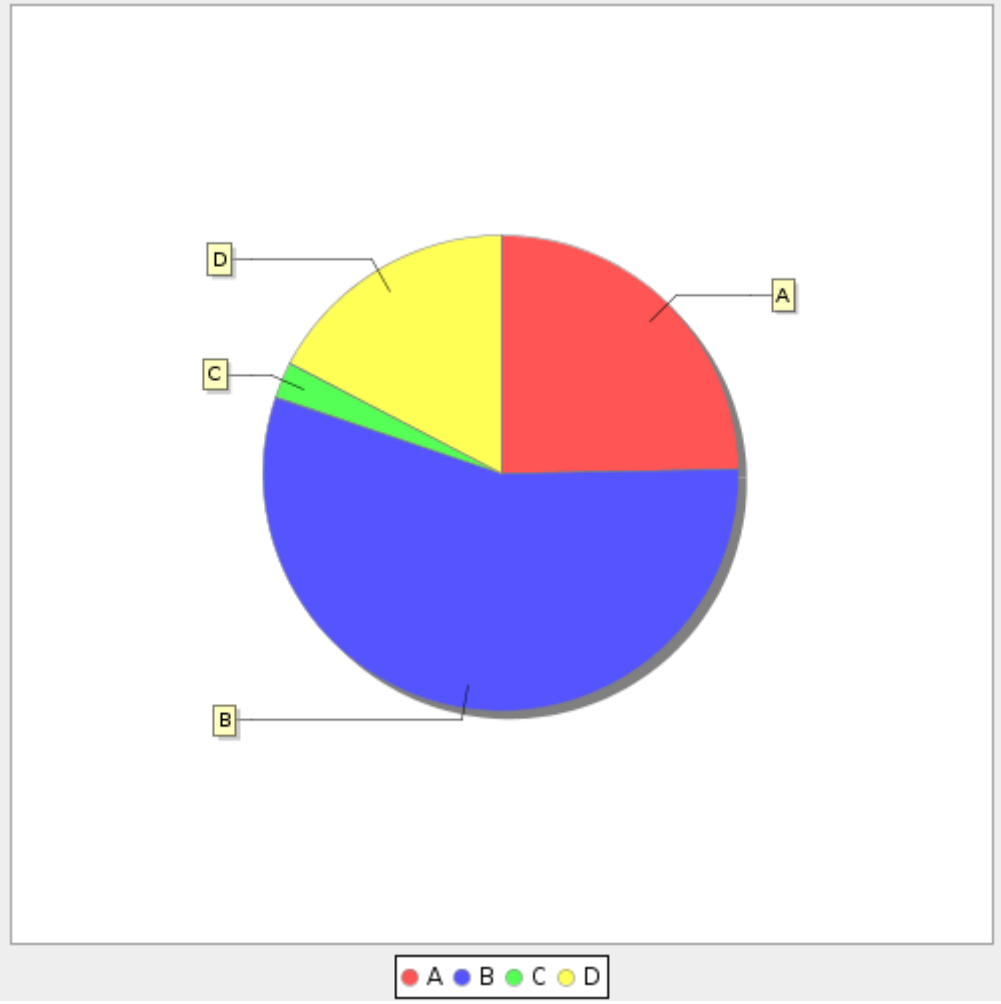
Show a chart in a GUI

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



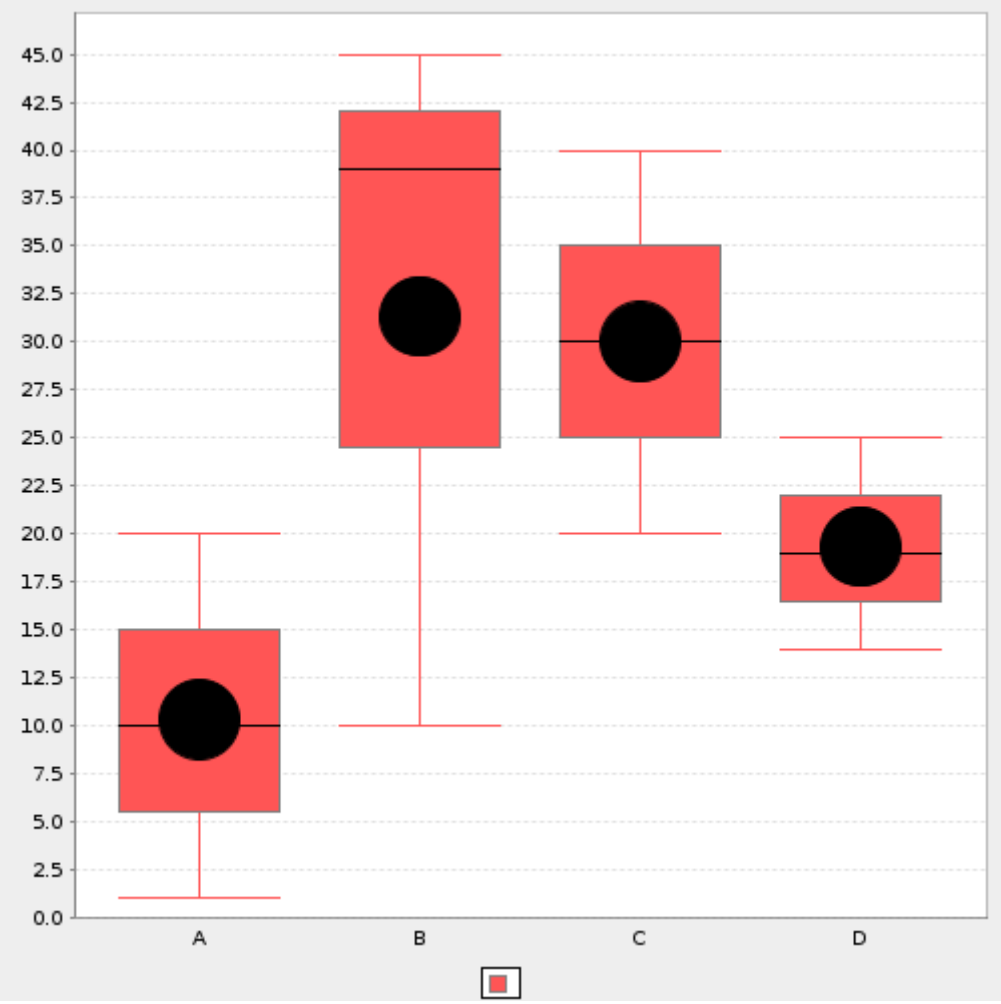
Get an image from a chart

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



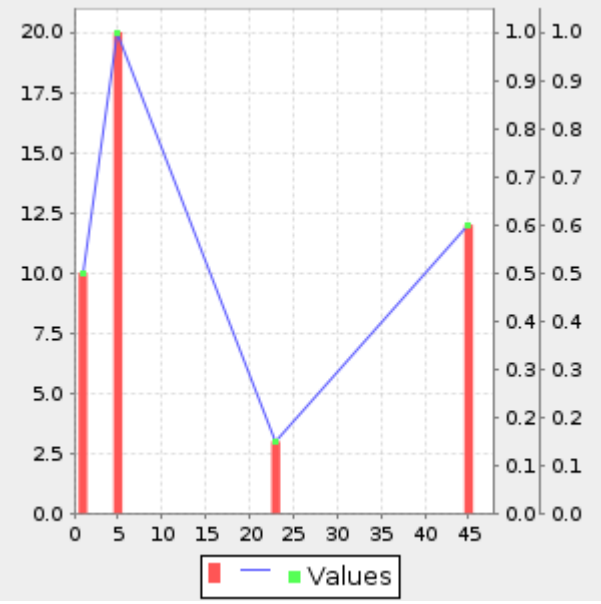
Save a chart to a file

```
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

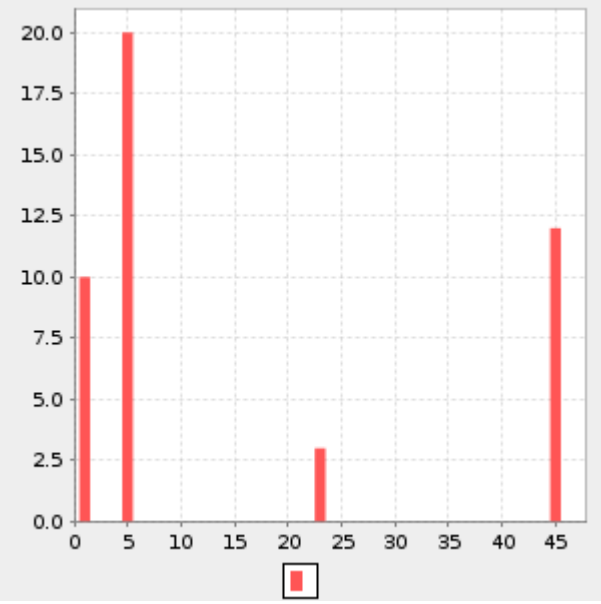
```
List data = [
        [1,10],[45,12],[23,3],[5,20]
]
Chart chart1 = Bar.xy(data)
Chart chart2 = Curve.curve(data)
Chart chart3 = Regression.linear(data)
chart1.overlay([chart2,chart3])
```



Creating Bar Charts

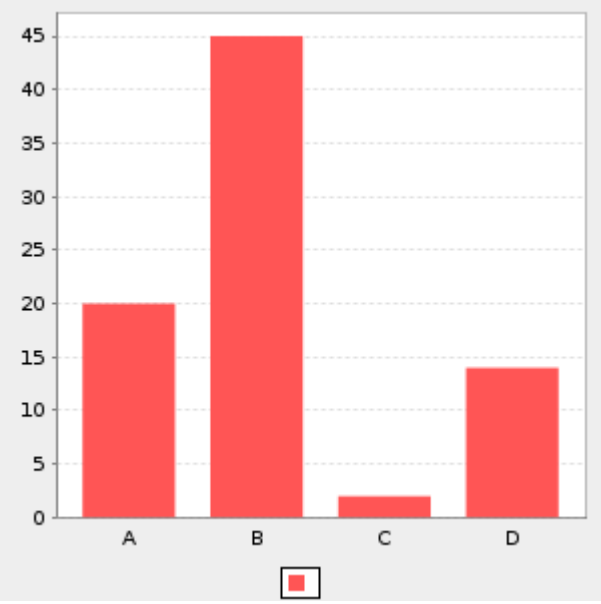
Create a basic bar chart

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



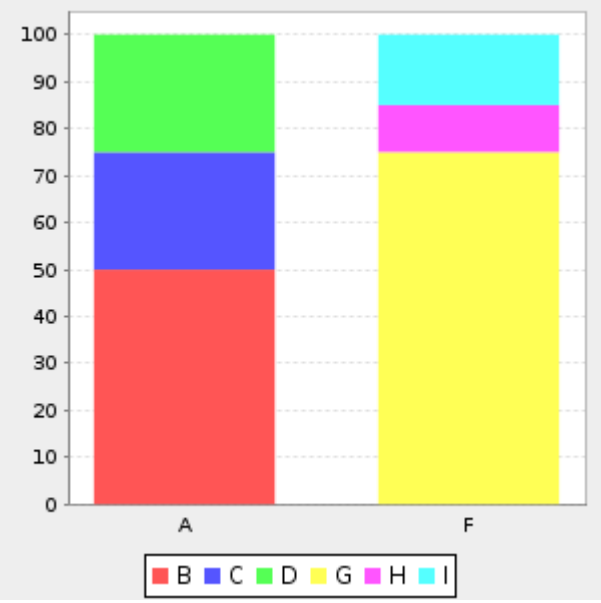
Create a bar chart with categories

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



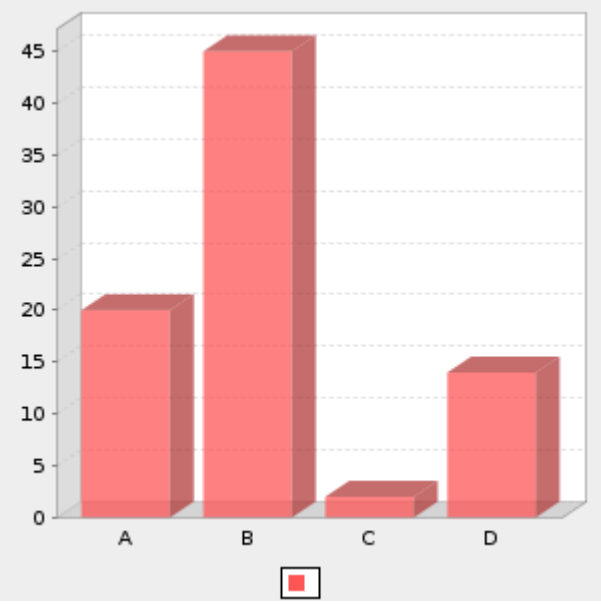
Create a stacked bar chart with two series of data

```
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

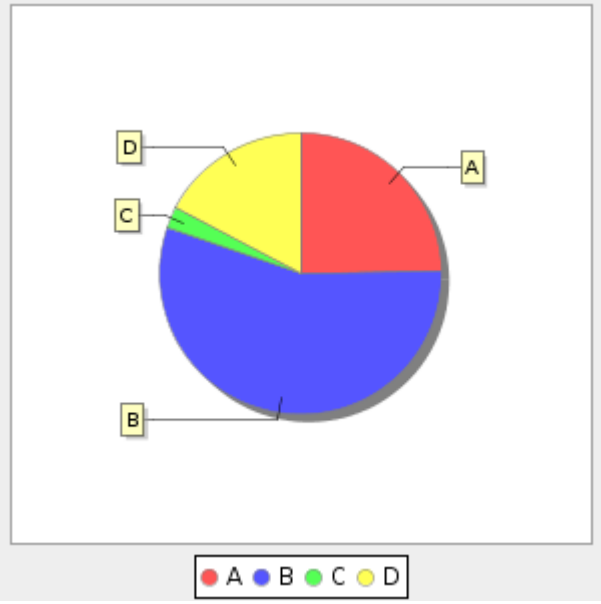
```
Map data = [
        "A":20,"B":45,"C":2,"D":14
]
Chart chart = Bar.category(data, trid: true)
```



Creating Pie Charts

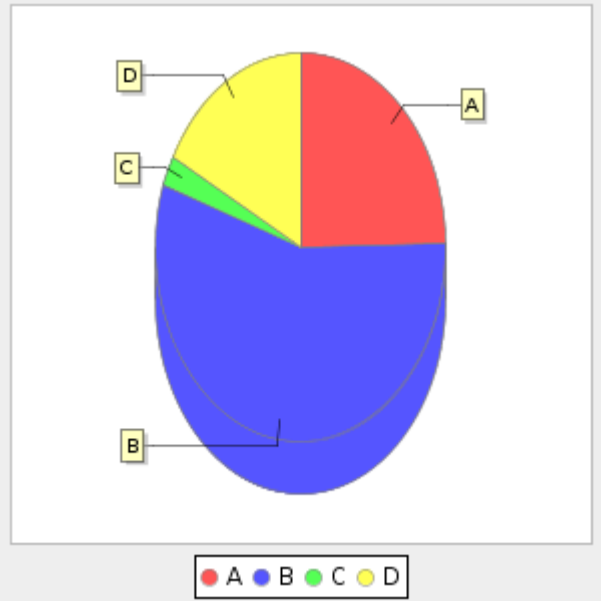
Create a pie chart

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



Create a 3D pie chart

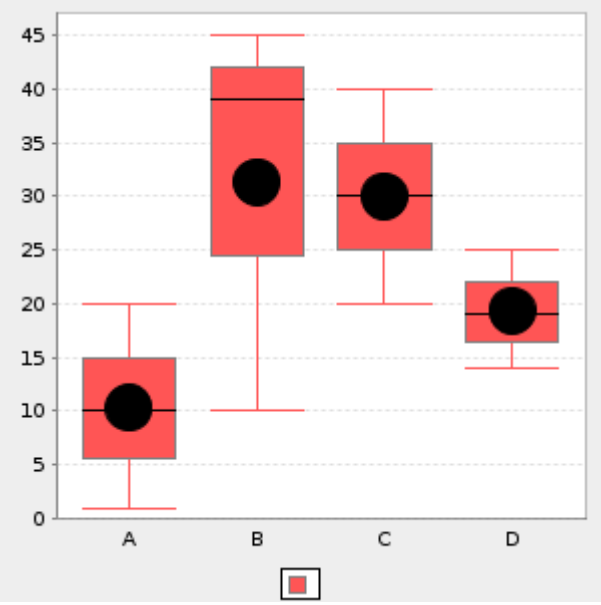
```
Map data = [
        "A":20,"B":45,"C":2,"D":14
]
Chart chart = Pie.pie(data, trid: true)
```



Creating Box Charts

Create a box chart

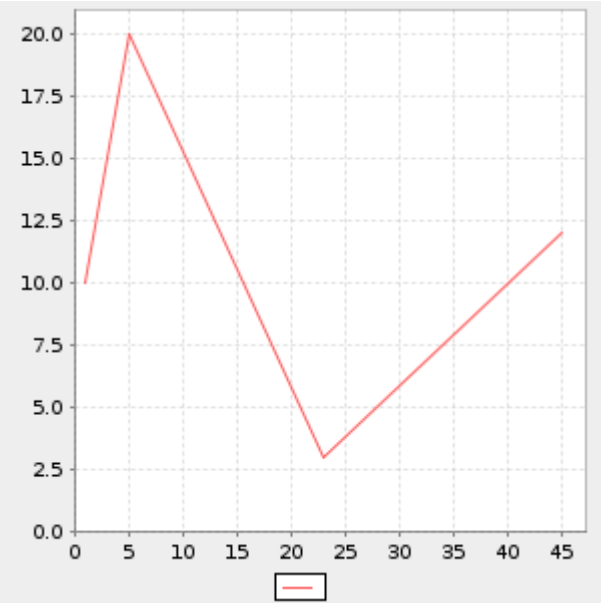
```
Map data = [
        "A":[1,10,20],
        "B":[45,39,10],
        "C":[40,30,20],
        "D":[14,25,19]
]
Chart chart = Box.box(data)
```



Creating Curve Charts

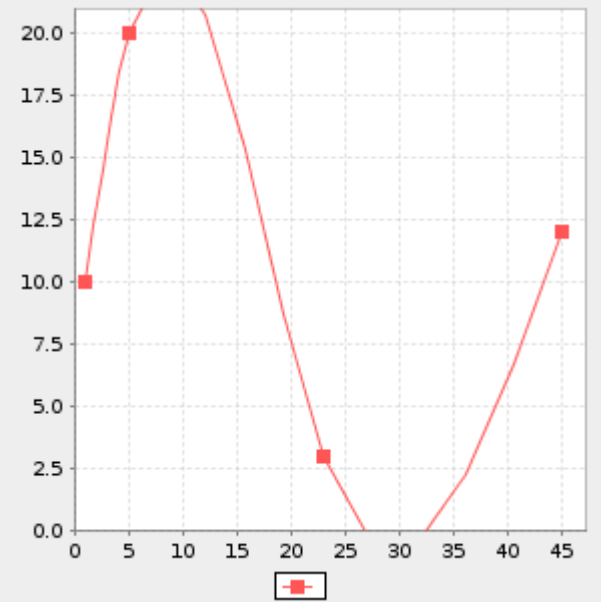
Create a curve chart

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



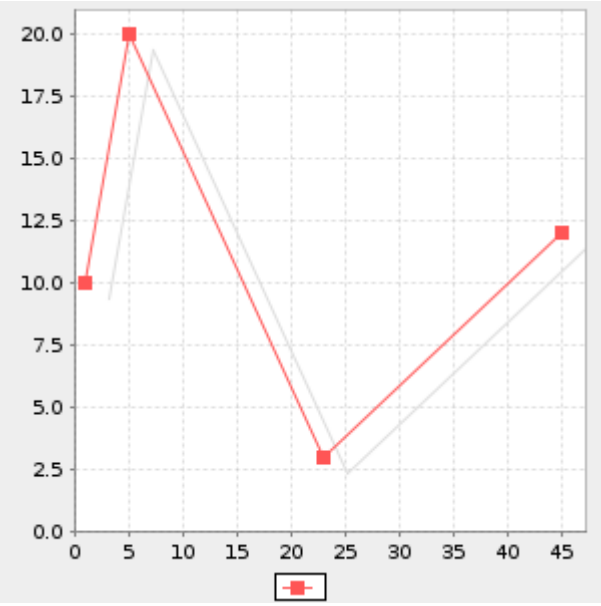
Create a smooth curve chart

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



Create a 3D curve chart

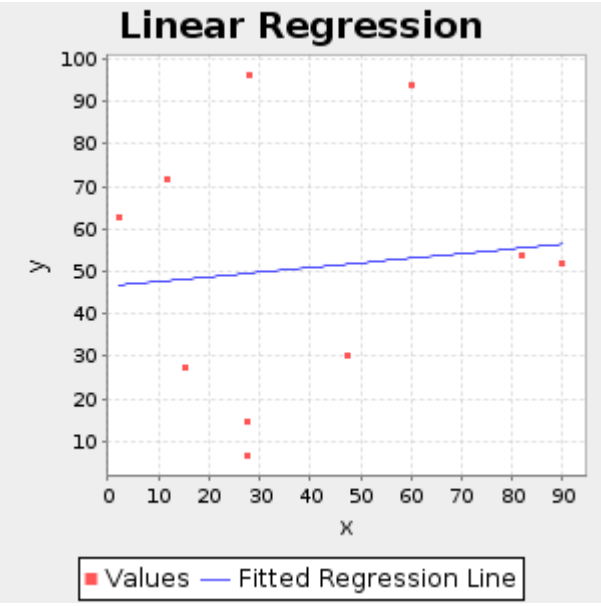
```
List data = [
        [1,10],[45,12],[23,3],[5,20]
]
Chart chart = Curve.curve(data, trid: true)
```



Creating Regression Charts

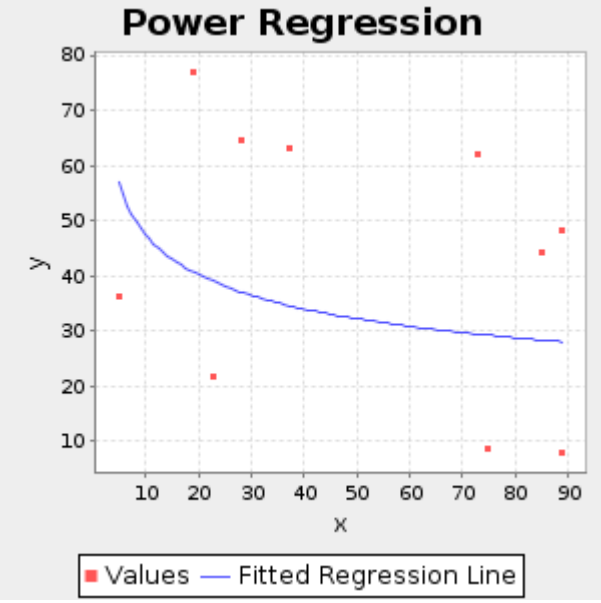
Create a linear regression chart

```
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

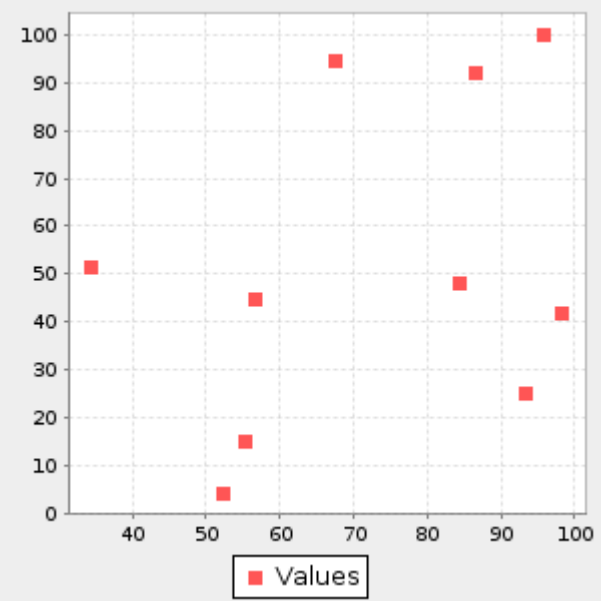
```
MultiPoint mulitPoint = Geometry.createRandomPoints(new Bounds(0,0,100,100).geometry,
10)
List data = mulitPoint.geometries.collect{ Point pt ->
    [pt.x, pt.y]
}
Chart chart = Regression.power(data)
```



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

The Feature classes are in the **geoscript.feature** package.

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-5b6b299_1620e77d8c3_-7ffc geom: null, id: null, name: null
```

Reading and Writing Schemas

The Schema IO classes are in the **geoscript.feature.io** package.

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

The Feature IO classes are in the **geoscript.feature.io** package.

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 Mar 10 05:52:38 UTC
2018</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-5b6b299_1620e77d8c3_-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 Mar 10 05:52:38 UTC
2018</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-5b6b299_1620e77d8c3_-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-5b6b299_1620e77d8c3_-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-5b6b299_1620e77d8c3_-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

The Filter classes are in the **geoscript.filter** package.

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

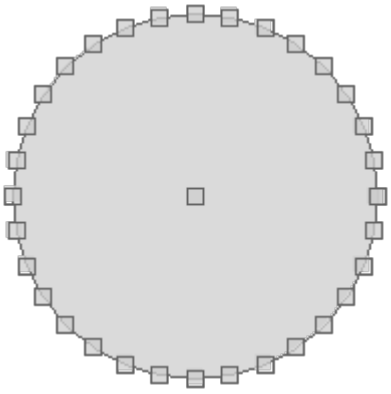
```
List<String> functionNames = Function.getFunctionNames()
println "There are ${functionNames.size()} Functions:"
functionNames.sort().subList(0,10).each { String name ->
    println name
}
```

```
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

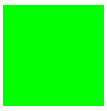
```
Function lowerCaseFunction = new Function("strToLowerCase(name)")
String lowerCaseName = lowerCaseFunction.evaluate(feature)
println lowerCaseName
```

```
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]
```

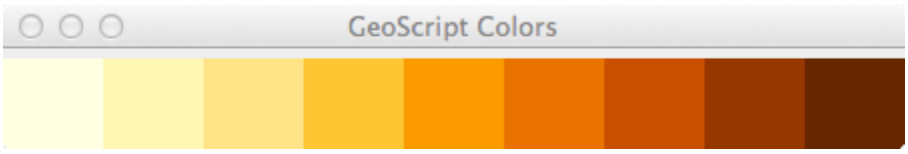
Displaying Colors

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

The Process classes are in the **geoscript.process** package.

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]
```

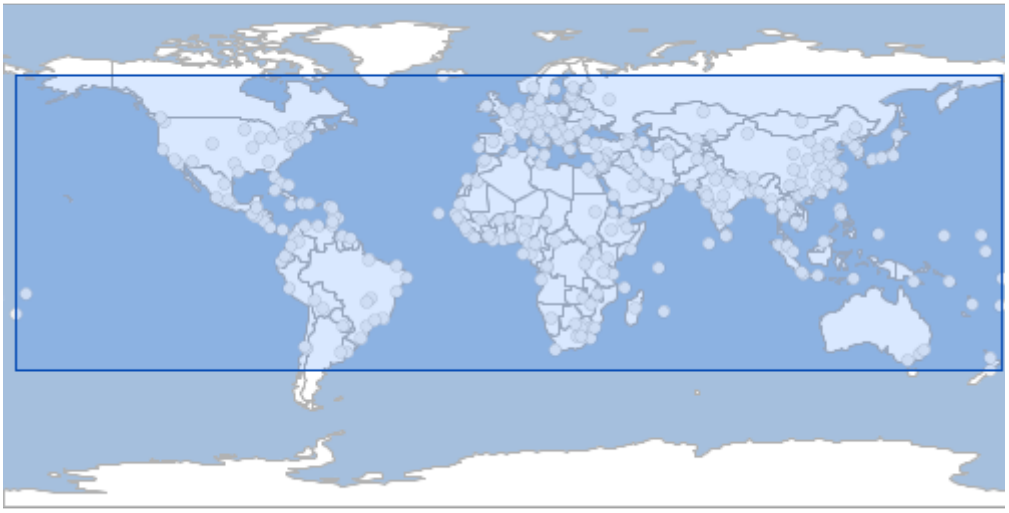
Get the output parameters

```
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
}
```

```
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:isValid
geo:buffer
geo:union
geo:intersection
geo:pointN
geo:difference
geo:area
geo:numGeometries
geo:symDifference
geo:isClosed
geo:convexHull
geo:crosses
geo:distance
geo:boundary
geo:centroid
geo:interiorPoint
geo:getGeometryN
geo:reproject
geo:equalsExactTolerance
geo:touches
geo:within
geo:simplify
geo:densify
geo:numPoints
geo:isSimple
geo:isWithinDistance
geo:overlaps
geo:relate
geo:dimension
geo:exteriorRing
geo:numInteriorRing
geo:geometryType
geo:envelope
geo:getX
```

```
geo:getY
geo:polygonize
geo:isRing
geo:startPoint
geo:endPoint
geo:equalsExact
geo:splitPolygon
geo:interiorRingN
geo:relatePattern
geo:length
geo:isEmpty
geo:contains
geo:disjoint
geo:intersects
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
```

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]
```

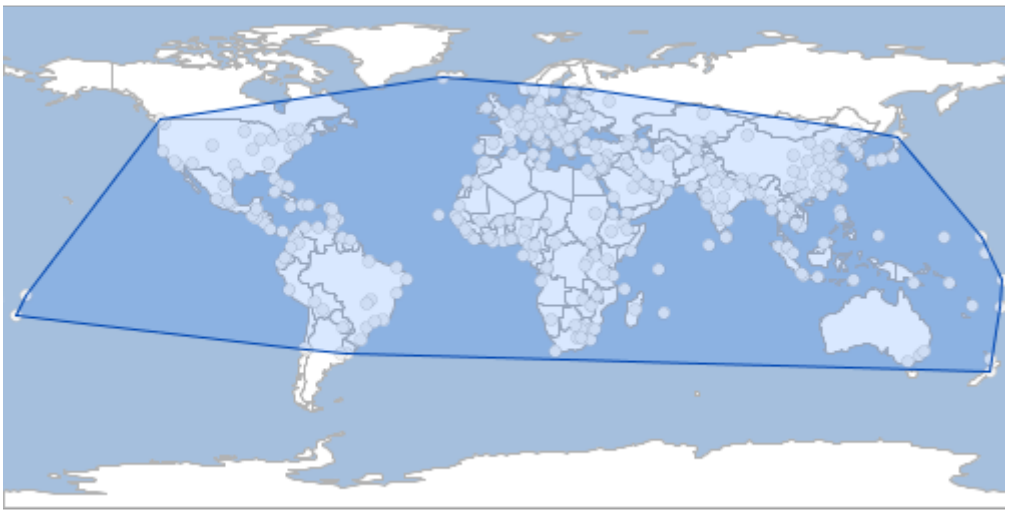
Get the output parameters

```
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
```



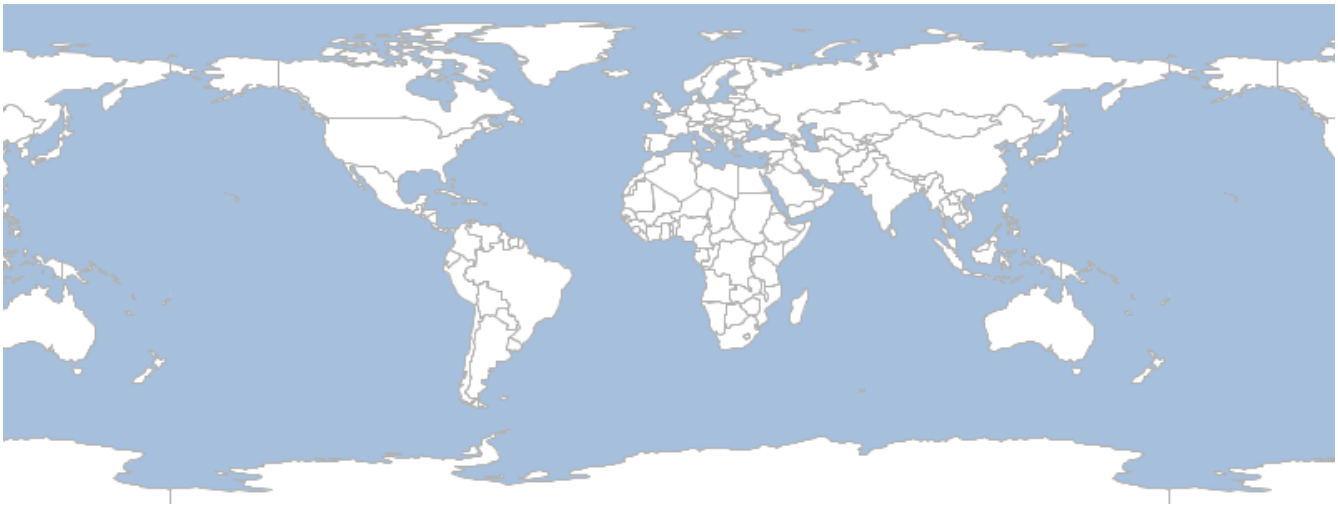
Render Recipes

The Render classes are in the **geoscript.render** package.

Creating Maps

Create a Map with Layers

```
Workspace workspace = new GeoPackage('src/main/resources/data.gpkg')
Layer countries = workspace.get("countries")
countries.style = new Fill("#ffffff") + new Stroke("#b2b2b2", 0.5)
Layer ocean = workspace.get("ocean")
ocean.style = new Fill("#a5bfdd")
Map map = new Map(
    width: 800,
    height: 300,
    layers: [ocean, countries]
)
File file = new File("map.png")
map.render(file)
```



Rendering Maps

Finding Renderers

Get all Renderers

```
List<Renderer> renderers = Renderers.list()
renderers.each { Renderer renderer ->
    println renderer.class.simpleName
}
```

```
ASCII
Base64
GeoTIFF
GIF
JPEG
Pdf
PNG
Svg
```

Get a Renderer

```
Renderer renderer = Renderers.find("png")
println renderer.class.simpleName
```

```
PNG
```

Image

Render a Map to an image using an Image Renderer

```
Workspace workspace = new GeoPackage('src/main/resources/data.gpkg')
Layer countries = workspace.get("countries")
countries.style = new Fill("#ffffff") + new Stroke("#b2b2b2", 0.5)
Layer ocean = workspace.get("ocean")
ocean.style = new Fill("#a5bfdd")
Map map = new Map(
        width: 800,
        height: 300,
        layers: [ocean, countries]
)
Image png = new Image("png")
BufferedImage image = png.render(map)
```

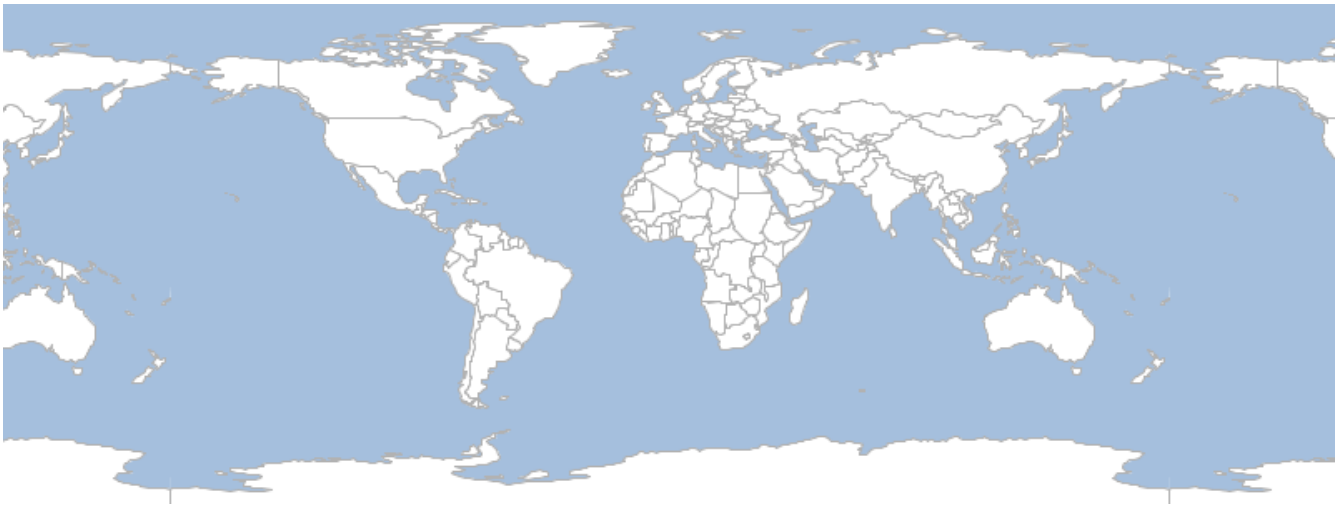
Render a Map to an OutputStream using the Image Renderer

```
Workspace workspace = new GeoPackage('src/main/resources/data.gpkg')
Layer countries = workspace.get("countries")
countries.style = new Fill("#ffffff") + new Stroke("#b2b2b2", 0.5)
Layer ocean = workspace.get("ocean")
ocean.style = new Fill("#a5bfdd")
Map map = new Map(
        width: 800,
        height: 300,
        layers: [ocean, countries]
)
Image jpeg = new Image("jpeg")
File file = new File("map.jpeg")
FileOutputStream out = new FileOutputStream(file)
jpeg.render(map, out)
out.close()
```

PNG

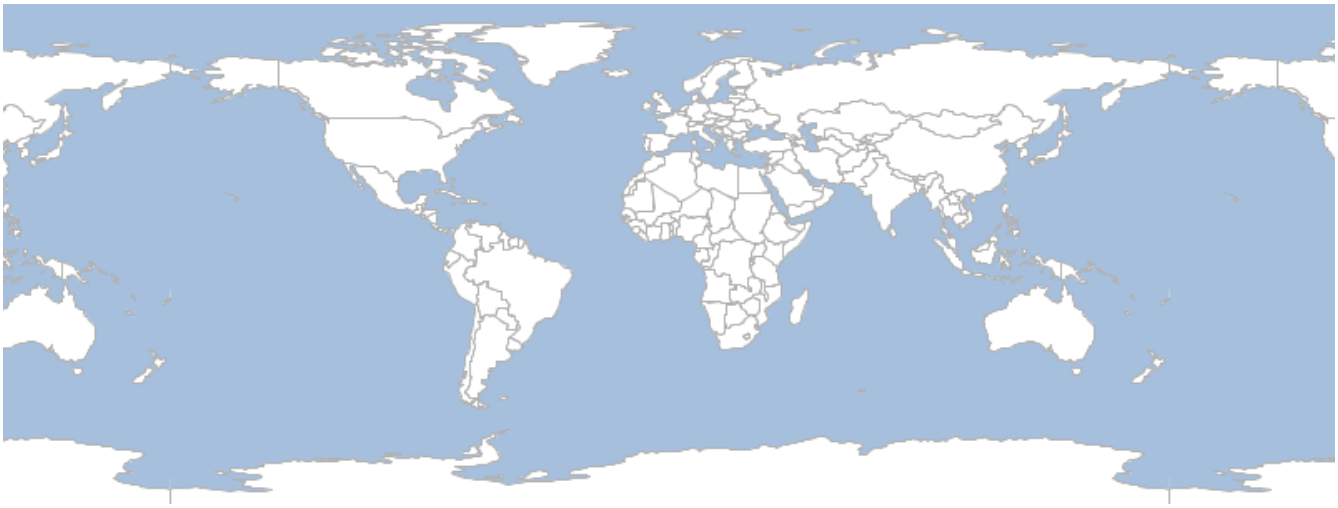
Render a Map to an Image using the PNG Renderer

```
Workspace workspace = new GeoPackage('src/main/resources/data.gpkg')
Layer countries = workspace.get("countries")
countries.style = new Fill("#ffffff") + new Stroke("#b2b2b2", 0.5)
Layer ocean = workspace.get("ocean")
ocean.style = new Fill("#a5bfdd")
Map map = new Map(
    width: 800,
    height: 300,
    layers: [ocean, countries]
)
PNG png = new PNG()
BufferedImage image = png.render(map)
```



Render a Map to an OutputStream using the PNG Renderer

```
Workspace workspace = new GeoPackage('src/main/resources/data.gpkg')
Layer countries = workspace.get("countries")
countries.style = new Fill("#ffffff") + new Stroke("#b2b2b2", 0.5)
Layer ocean = workspace.get("ocean")
ocean.style = new Fill("#a5bfdd")
Map map = new Map(
    width: 800,
    height: 300,
    layers: [ocean, countries]
)
PNG png = new PNG()
File file = new File("map.png")
FileOutputStream out = new FileOutputStream(file)
png.render(map, out)
out.close()
```



JPEG

Render a Map to an Image using the JPEG Renderer

```
Workspace workspace = new GeoPackage('src/main/resources/data.gpkg')
Layer countries = workspace.get("countries")
countries.style = new Fill("#ffffff") + new Stroke("#b2b2b2", 0.5)
Layer ocean = workspace.get("ocean")
ocean.style = new Fill("#a5bfdd")
Map map = new Map(
        width: 800,
        height: 300,
        layers: [ocean, countries]
)
JPEG jpeg = new JPEG()
BufferedImage image = jpeg.render(map)
```

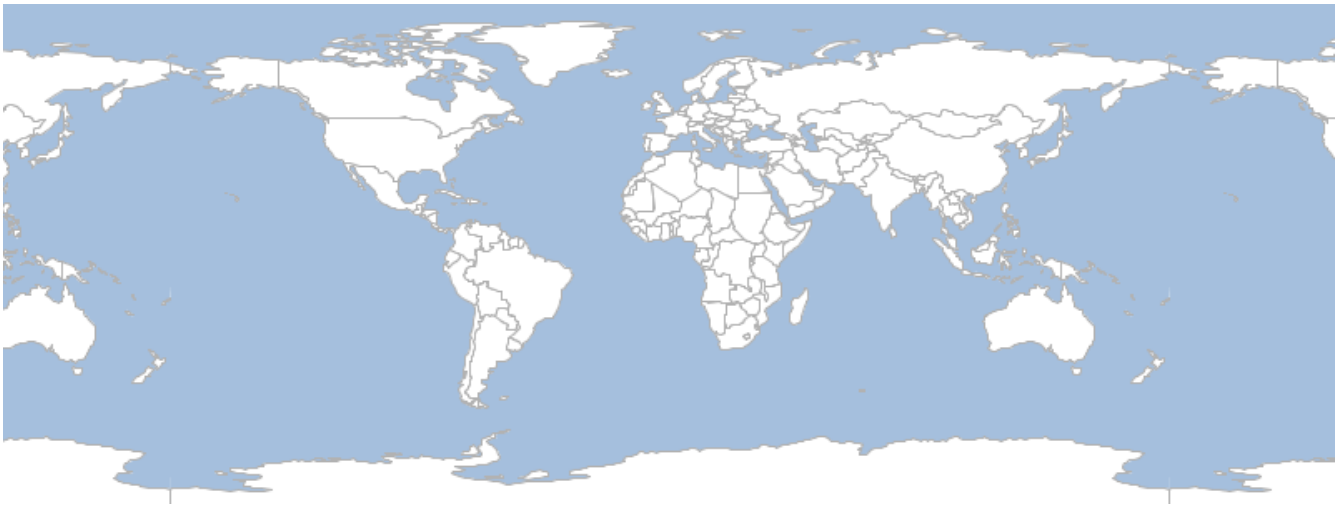
Render a Map to an OutputStream using the JPEG Renderer

```
Workspace workspace = new GeoPackage('src/main/resources/data.gpkg')
Layer countries = workspace.get("countries")
countries.style = new Fill("#ffffff") + new Stroke("#b2b2b2", 0.5)
Layer ocean = workspace.get("ocean")
ocean.style = new Fill("#a5bfdd")
Map map = new Map(
        width: 800,
        height: 300,
        layers: [ocean, countries]
)
JPEG jpeg = new JPEG()
File file = new File("map.jpeg")
FileOutputStream out = new FileOutputStream(file)
jpeg.render(map, out)
out.close()
```

GIF

Render a Map to an Image using the GIF Renderer

```
Workspace workspace = new GeoPackage('src/main/resources/data.gpkg')
Layer countries = workspace.get("countries")
countries.style = new Fill("#ffffff") + new Stroke("#b2b2b2", 0.5)
Layer ocean = workspace.get("ocean")
ocean.style = new Fill("#a5bfdd")
Map map = new Map(
        width: 800,
        height: 300,
        layers: [ocean, countries]
)
GIF gif = new GIF()
BufferedImage image = gif.render(map)
```

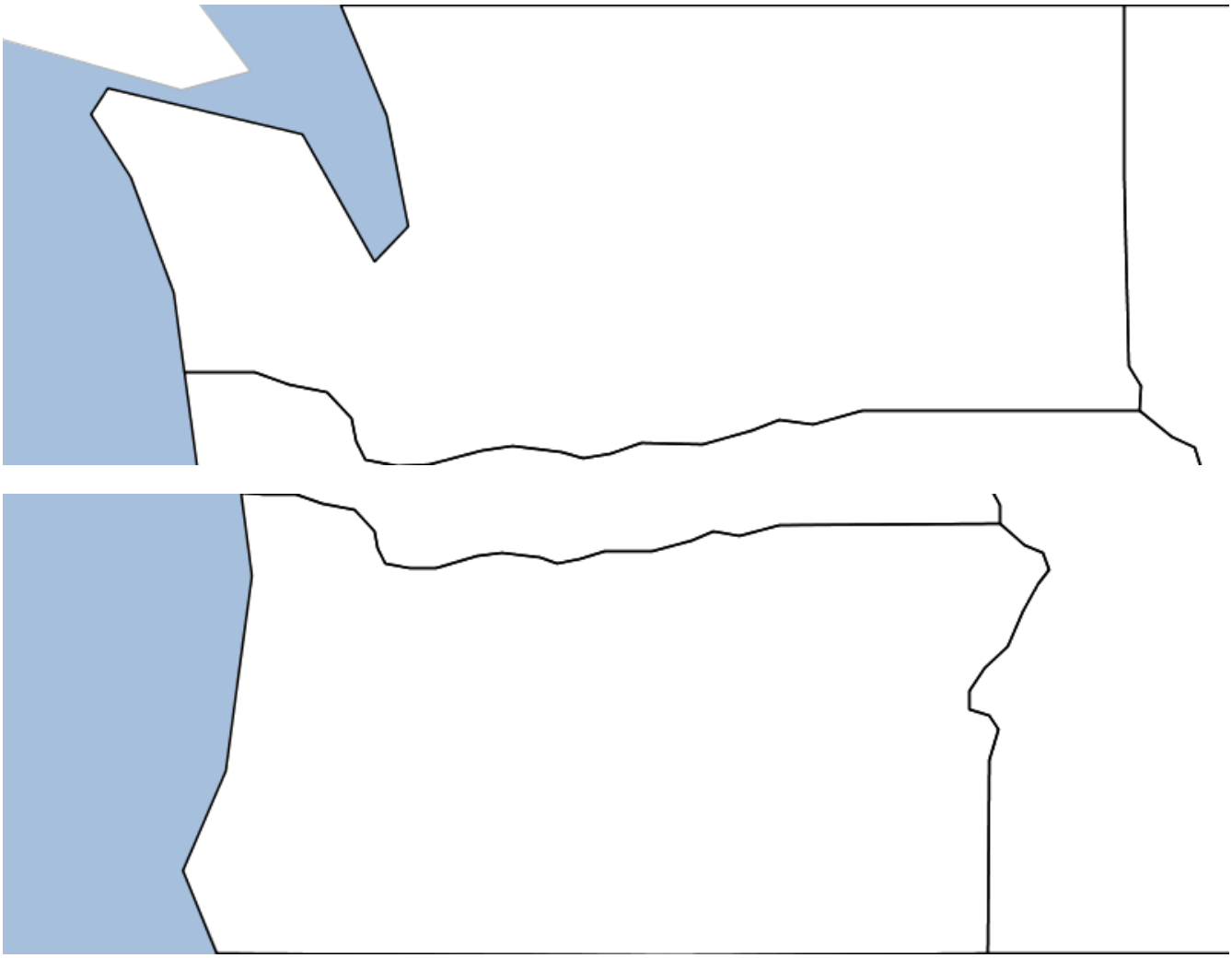


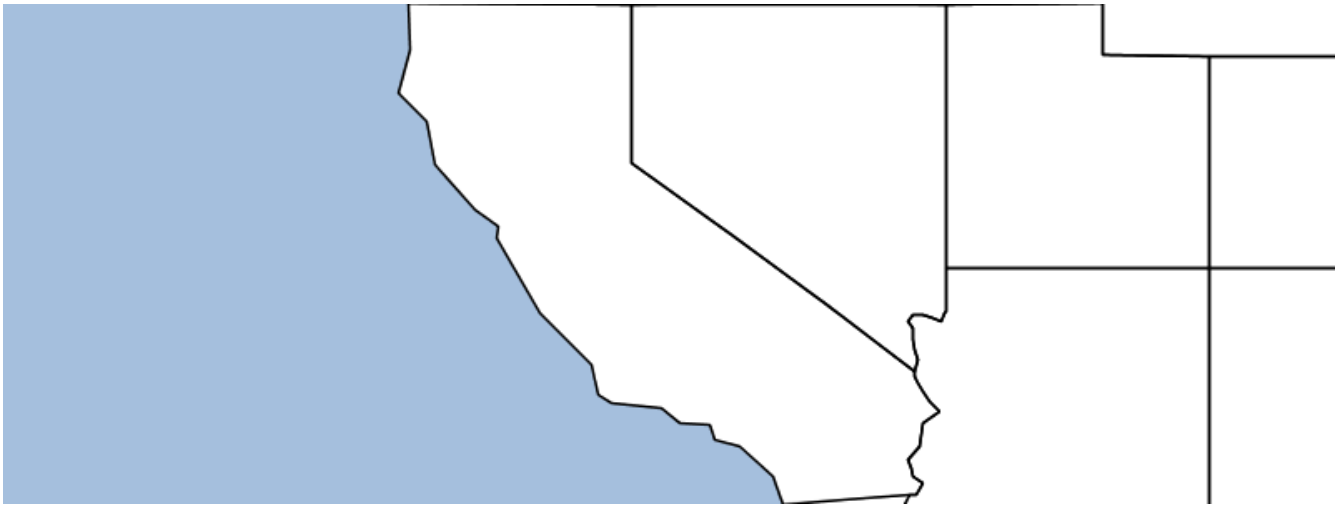
Render a Map to an OutputStream using the GIF Renderer

```
Workspace workspace = new GeoPackage('src/main/resources/data.gpkg')
Layer countries = workspace.get("countries")
countries.style = new Fill("#ffffff") + new Stroke("#b2b2b2", 0.5)
Layer ocean = workspace.get("ocean")
ocean.style = new Fill("#a5bfdd")
Map map = new Map(
        width: 800,
        height: 300,
        layers: [ocean, countries]
)
GIF gif = new GIF()
File file = new File("map.gif")
gif.render(map, new FileOutputStream(file))
```

Render a Map to an animated GIF using the GIF Renderer

```
Workspace workspace = new GeoPackage('src/main/resources/data.gpkg')
Layer states = workspace.get("states")
states.style = new Fill("") + new Stroke("black", 1.0)
Layer countries = workspace.get("countries")
countries.style = new Fill("#ffffff") + new Stroke("#b2b2b2", 0.5)
Layer ocean = workspace.get("ocean")
ocean.style = new Fill("#a5bfdd")
Map map = new Map(
        width: 800,
        height: 300,
        layers: [ocean, countries, states]
)

GIF gif = new GIF()
List images = ["Washington","Oregon","California"].collect { String state ->
    map.bounds = states.getFeatures("NAME_1 = '${state}'")[0].bounds
    def image = gif.render(map)
    image
}
File file = new File("states.gif")
gif.renderAnimated(images, file, 500, true)
```

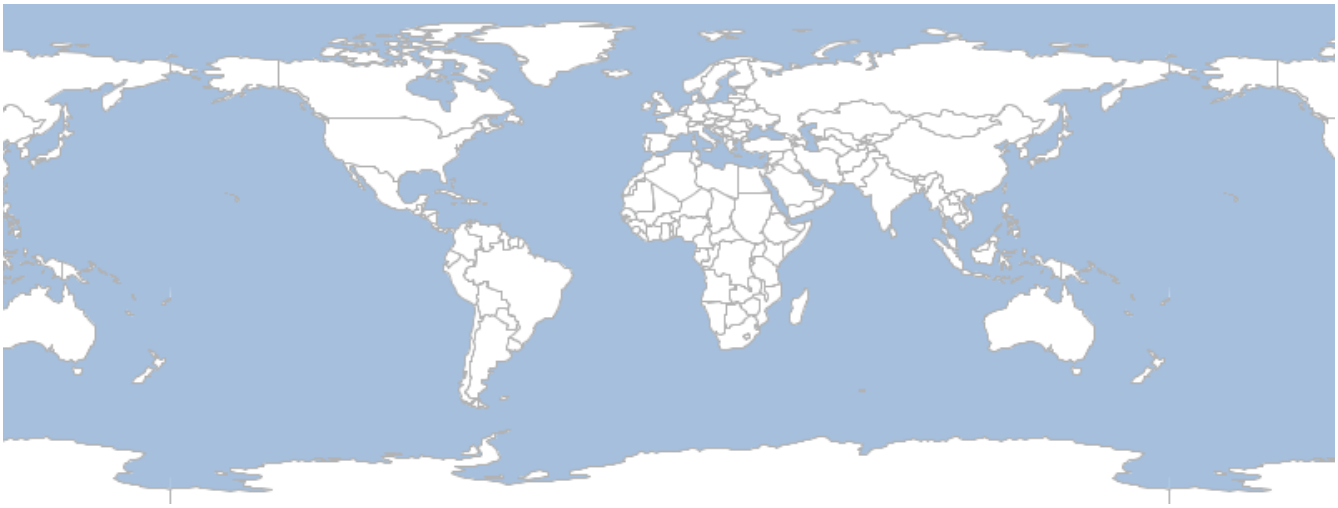




GeoTIFF

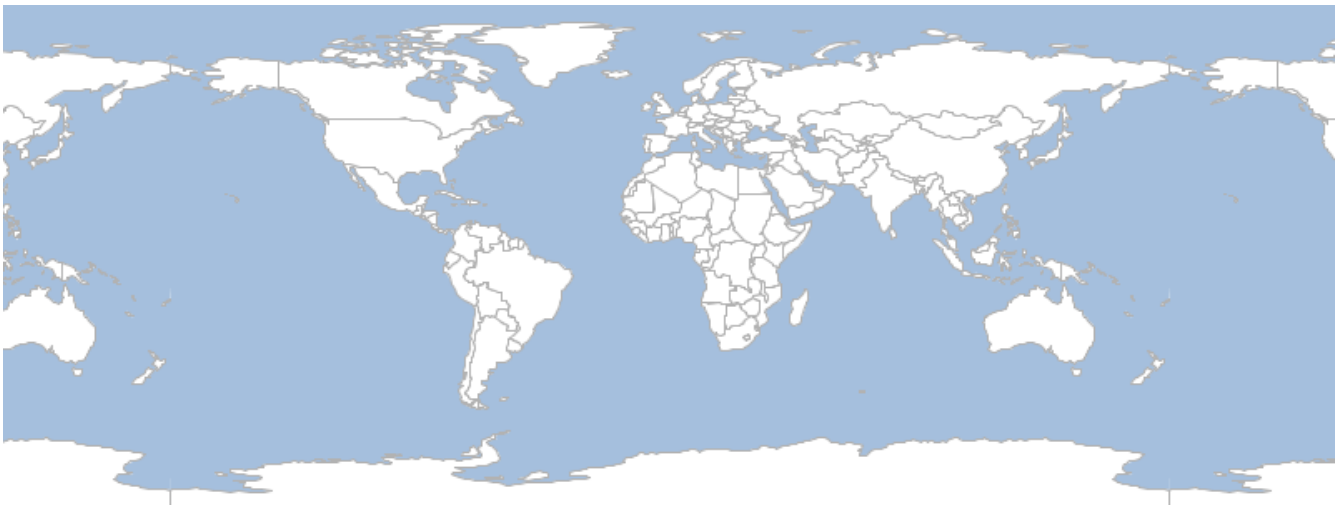
Render a Map to an Image using the GeoTIFF Renderer

```
Workspace workspace = new GeoPackage('src/main/resources/data.gpkg')
Layer countries = workspace.get("countries")
countries.style = new Fill("#ffffff") + new Stroke("#b2b2b2", 0.5)
Layer ocean = workspace.get("ocean")
ocean.style = new Fill("#a5bfdd")
Map map = new Map(
        width: 800,
        height: 300,
        layers: [ocean, countries]
)
GeoTIFF geotiff = new GeoTIFF()
BufferedImage image = geotiff.render(map)
```



Render a Map to an OutputStream using the GeoTIFF Renderer

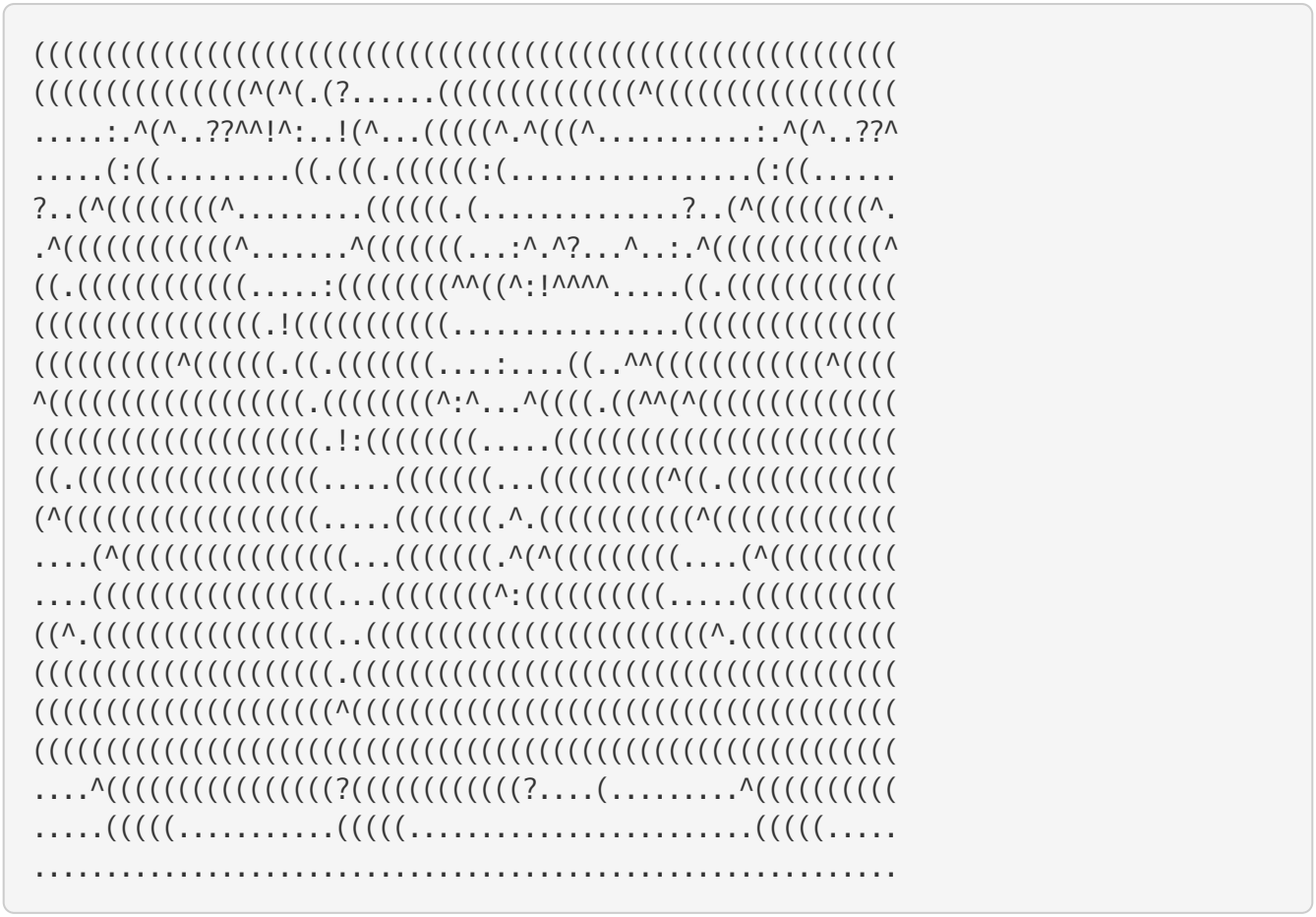
```
Workspace workspace = new GeoPackage('src/main/resources/data.gpkg')
Layer countries = workspace.get("countries")
countries.style = new Fill("#ffffff") + new Stroke("#b2b2b2", 0.5)
Layer ocean = workspace.get("ocean")
ocean.style = new Fill("#a5bfdd")
Map map = new Map(
        width: 800,
        height: 300,
        layers: [ocean, countries]
)
GeoTIFF geotiff = new GeoTIFF()
File file = new File("map.tif")
geotiff.render(map, new FileOutputStream(file))
```



ASCII

Render a Map to an string using the ASCII Renderer

```
Workspace workspace = new GeoPackage('src/main/resources/data.gpkg')
Layer countries = workspace.get("countries")
countries.style = new Fill("#ffffff") + new Stroke("#b2b2b2", 0.5)
Layer ocean = workspace.get("ocean")
ocean.style = new Fill("#a5bfdd")
Map map = new Map(
        width: 800,
        height: 300,
        layers: [ocean, countries]
)
ASCII ascii = new ASCII(width: 60)
String asciiStr = ascii.render(map)
println asciiStr
```



Render a Map to an text file using the ASCII Renderer

```
Workspace workspace = new GeoPackage('src/main/resources/data.gpkg')
Layer countries = workspace.get("countries")
countries.style = new Fill("#ffffff") + new Stroke("#b2b2b2", 0.5)
Layer ocean = workspace.get("ocean")
ocean.style = new Fill("#a5bfdd")
Map map = new Map(
        width: 800,
        height: 300,
        layers: [ocean, countries]
)
ASCII ascii = new ASCII(width: 60)
File file = new File("map.txt")
FileOutputStream out = new FileOutputStream(file)
ascii.render(map, out)
out.close()
```

```
((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((
((((((((((((((^(^(.(?.....((((((((((((((^((((((((((((((((
.....:.^(^..??^^!^:..!(^...(((((^.^(((^...........:.^(^..??^
.....(:((.........((.(((.((((((:(.................(:((......
?..(^(((((((^.........((((((.(..............?..(^(((((((^.
.^((((((((((^.......^(((((((...:^.^?...^..:.^((((((((((^
((.((((((((((.....:(((((((^^((^:!^^^^.....((.((((((((((
((((((((((((((.!((((((((((.................((((((((((((
(((((((((^((((((.((.(((((((....:....((..^^((((((((((^((((
^(((((((((((((((((.(((((((^:^...^((((.((^^(^((((((((((((
((((((((((((((((((.!:(((((((.....((((((((((((((((((((((
((.((((((((((((((((.....(((((((...(((((((((^((.((((((((((
(^((((((((((((((((.....(((((((.^.((((((((((^((((((((((((
....(^((((((((((((((...(((((((.^(^(((((((((....(^(((((((((
....((((((((((((((...(((((((^:((((((((((.....((((((((((
((^.((((((((((((((..(((((((((((((((((((((^.((((((((((
((((((((((((((((((.((((((((((((((((((((((((((((((((((((
((((((((((((((((((^((((((((((((((((((((((((((((((((((((
((((((((((((((((((((((((((((((((((((((((((((((((((((((
....^((((((((((((((?((((((((((?....(.........^((((((((((
.....(((((...........((((.......................(((((.....
..........................................................
```

Base64

Render a Map to an string using the Base64 Renderer

```
Workspace workspace = new GeoPackage('src/main/resources/data.gpkg')
Layer countries = workspace.get("countries")
countries.style = new Fill("#ffffff") + new Stroke("#b2b2b2", 0.5)
Layer ocean = workspace.get("ocean")
ocean.style = new Fill("#a5bfdd")
Map map = new Map(
        width: 800,
        height: 300,
        layers: [ocean, countries]
)
Base64 base64 = new Base64()
String base64Str = base64.render(map)
println base64Str
```

```
image/png;base64,iVBORw0KGgoAAAANSUhEUgAAAyAAAAEsC...
```

Render a Map to an text file using the Base64 Renderer

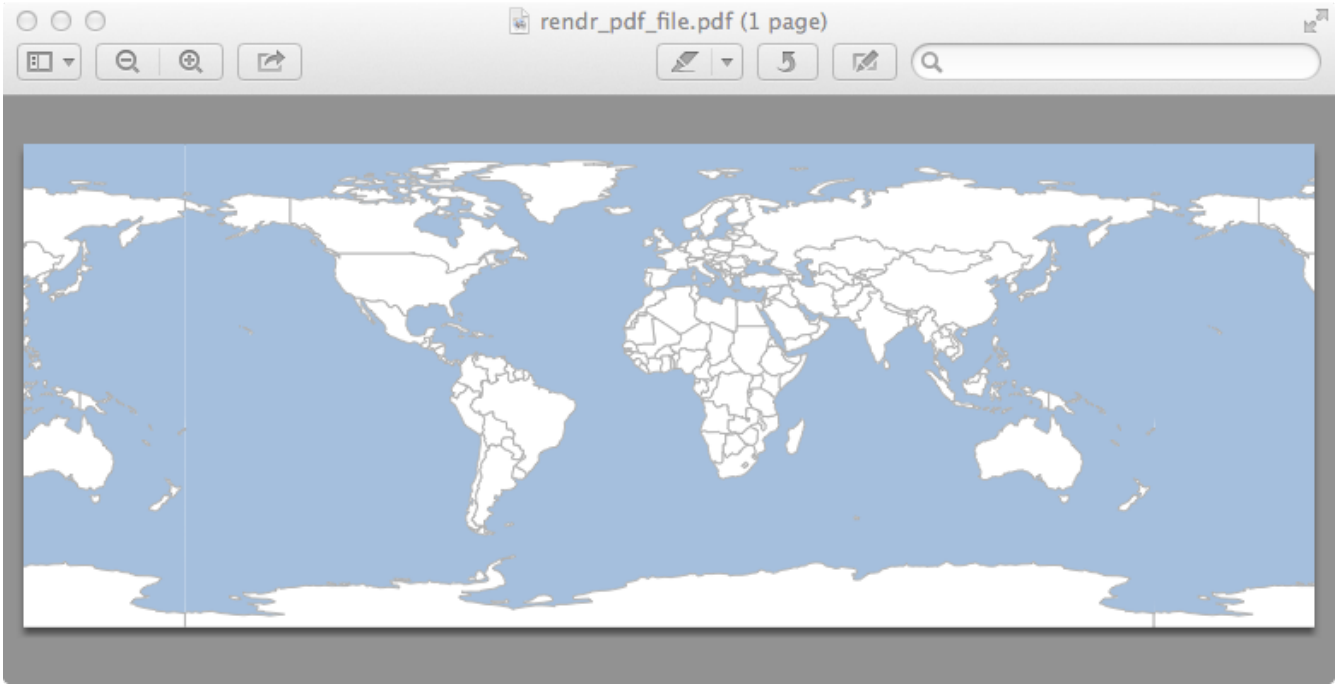
```
Workspace workspace = new GeoPackage('src/main/resources/data.gpkg')
Layer countries = workspace.get("countries")
countries.style = new Fill("#ffffff") + new Stroke("#b2b2b2", 0.5)
Layer ocean = workspace.get("ocean")
ocean.style = new Fill("#a5bfdd")
Map map = new Map(
        width: 800,
        height: 300,
        layers: [ocean, countries]
)
Base64 base64 = new Base64()
File file = new File("map.txt")
base64.render(map, new FileOutputStream(file))
```

```
iVBORw0KGgoAAAANSUhEUgAAAyAAAAEsCAYAAAA7Ldc6AACAAE...
```

PDF

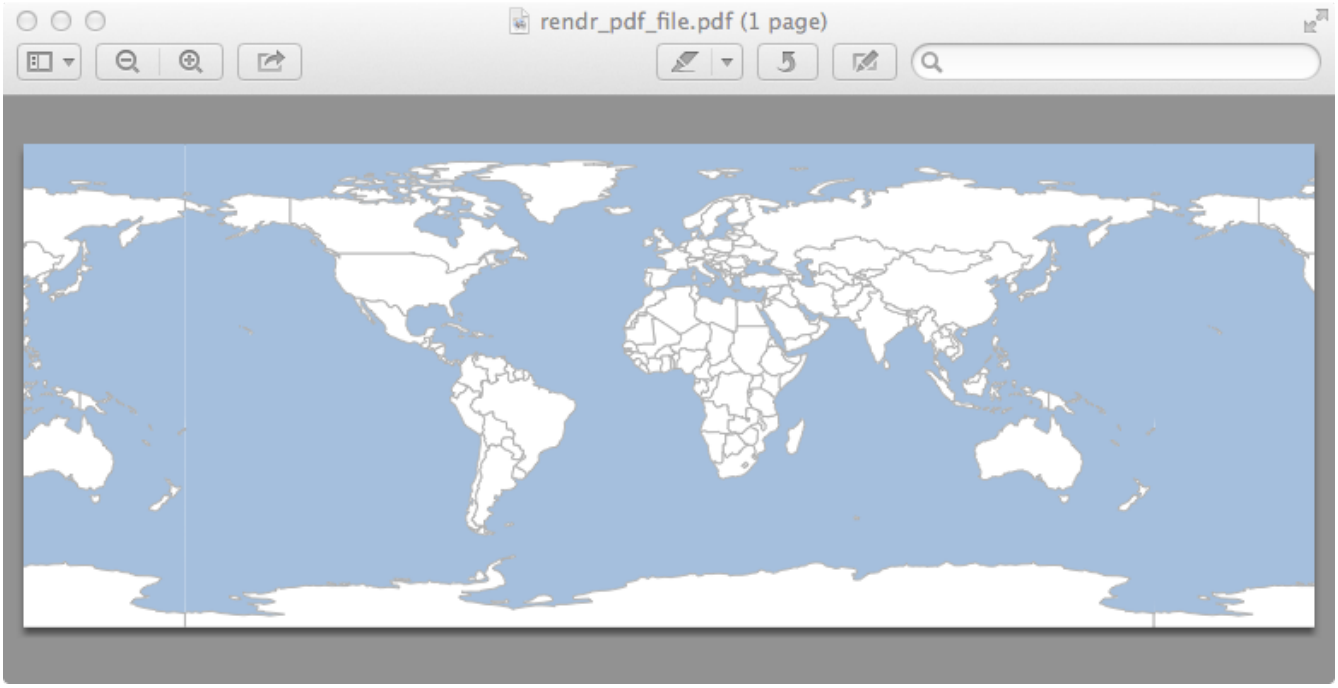
Render a Map to a PDF Document using the PDF Renderer

```
Workspace workspace = new GeoPackage('src/main/resources/data.gpkg')
Layer countries = workspace.get("countries")
countries.style = new Fill("#ffffff") + new Stroke("#b2b2b2", 0.5)
Layer ocean = workspace.get("ocean")
ocean.style = new Fill("#a5bfdd")
Map map = new Map(
        width: 800,
        height: 300,
        layers: [ocean, countries]
)
Pdf pdf = new Pdf()
com.lowagie.text.Document document = pdf.render(map)
```



Render a Map to a PDF file using the PDF Renderer

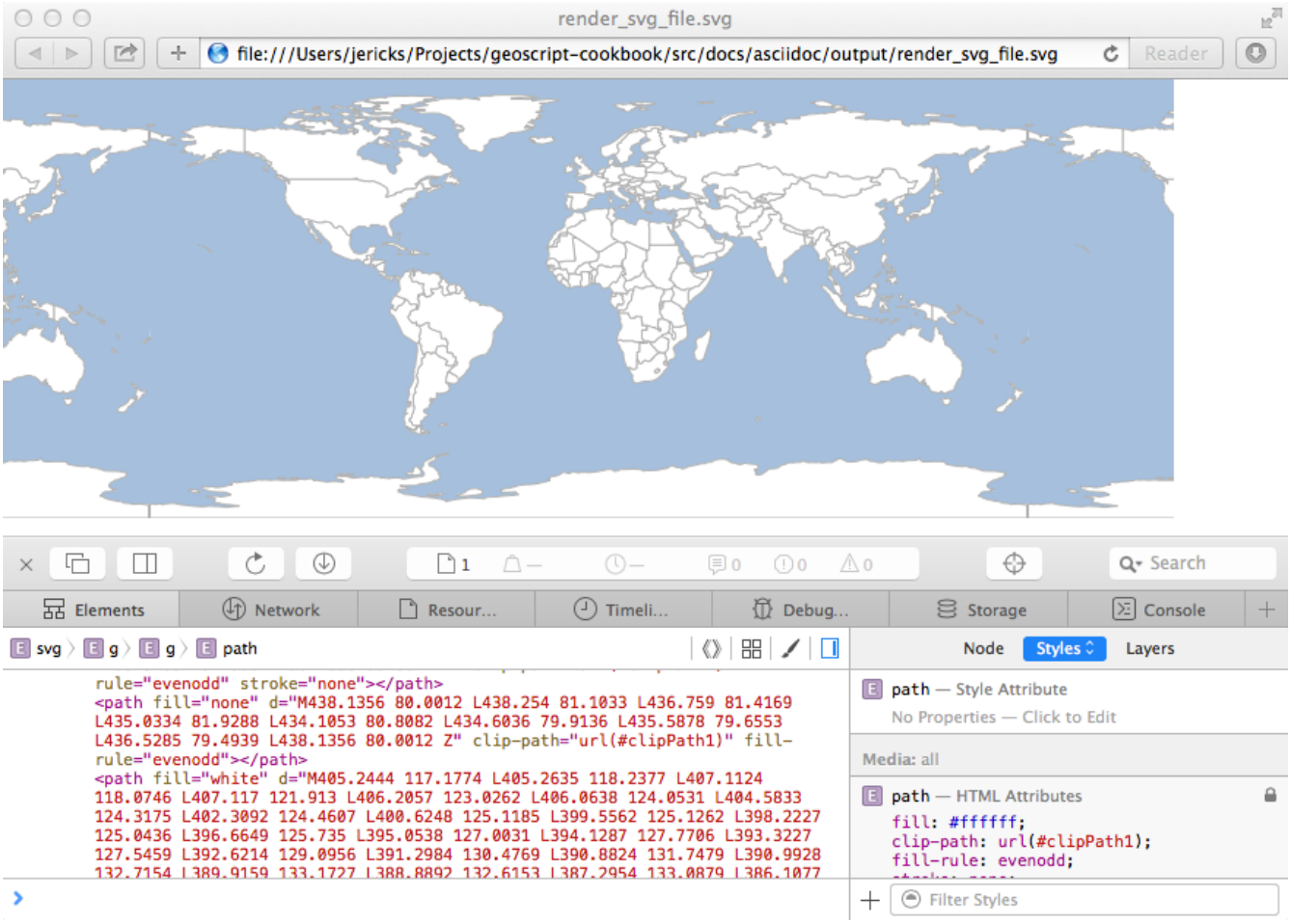
```
Workspace workspace = new GeoPackage('src/main/resources/data.gpkg')
Layer countries = workspace.get("countries")
countries.style = new Fill("#ffffff") + new Stroke("#b2b2b2", 0.5)
Layer ocean = workspace.get("ocean")
ocean.style = new Fill("#a5bfdd")
Map map = new Map(
        width: 800,
        height: 300,
        layers: [ocean, countries]
)
Pdf pdf = new Pdf()
File file = new File("map.pdf")
pdf.render(map, new FileOutputStream(file))
```



SVG

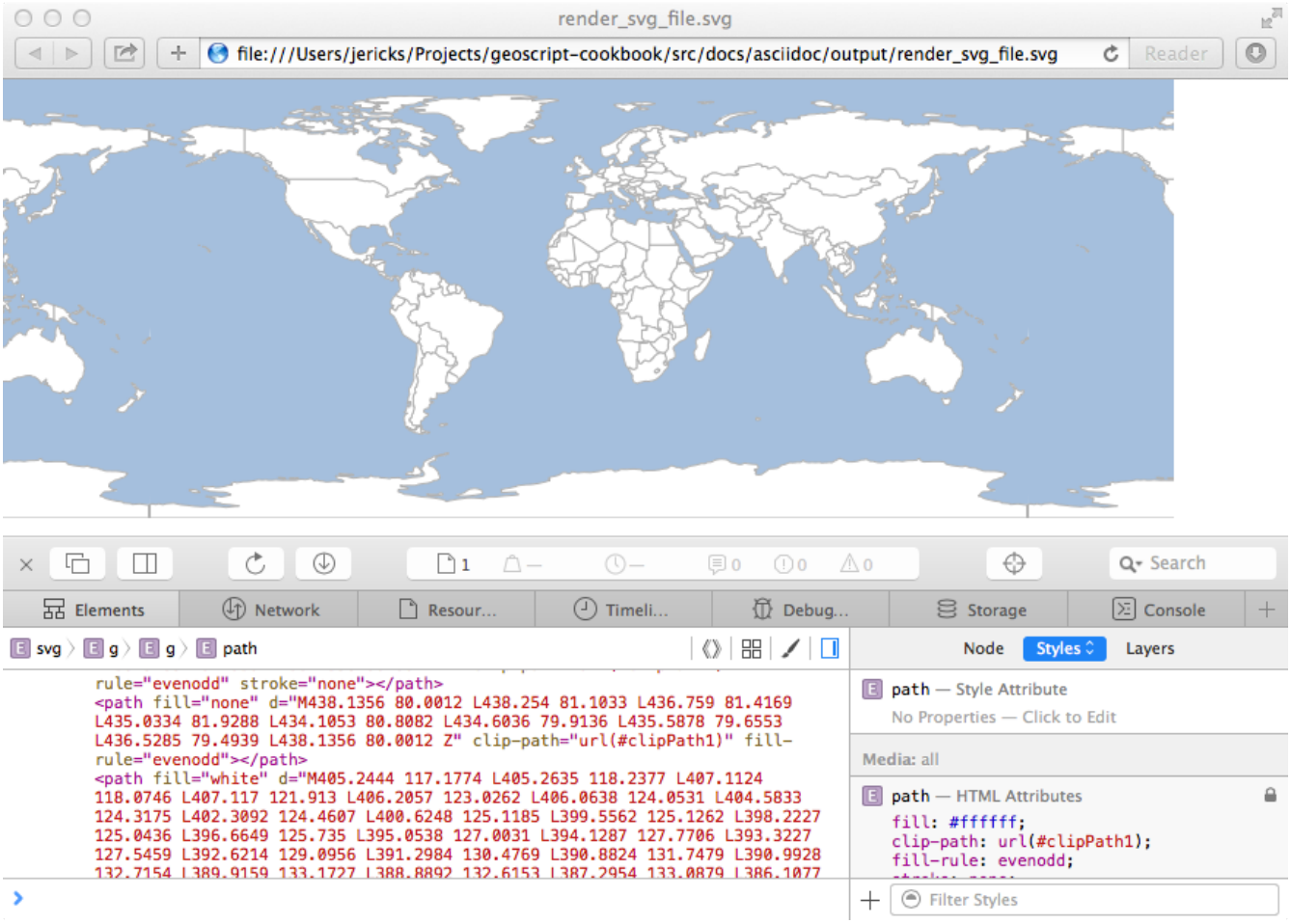
Render a Map to a SVG Document using the SVG Renderer

```
Workspace workspace = new GeoPackage('src/main/resources/data.gpkg')
Layer countries = workspace.get("countries")
countries.style = new Fill("#ffffff") + new Stroke("#b2b2b2", 0.5)
Layer ocean = workspace.get("ocean")
ocean.style = new Fill("#a5bfdd")
Map map = new Map(
        width: 800,
        height: 300,
        layers: [ocean, countries]
)
Svg svg = new Svg()
org.w3c.dom.Document document = svg.render(map)
```



Render a Map to a SVG file using the SVG Renderer

```
Workspace workspace = new GeoPackage('src/main/resources/data.gpkg')
Layer countries = workspace.get("countries")
countries.style = new Fill("#ffffff") + new Stroke("#b2b2b2", 0.5)
Layer ocean = workspace.get("ocean")
ocean.style = new Fill("#a5bfdd")
Map map = new Map(
        width: 800,
        height: 300,
        layers: [ocean, countries]
)
Svg svg = new Svg()
File file = new File("map.svg")
FileOutputStream out = new FileOutputStream(file)
svg.render(map, out)
out.close()
```



Displaying Maps

Finding Displayers

Get all Displayers

```
List<Displayer> displayers = Displayers.list()
displayers.each { Displayer displayer ->
    println displayer.class.simpleName
}
```

```
MapWindow
Window
```

Get a Displayer

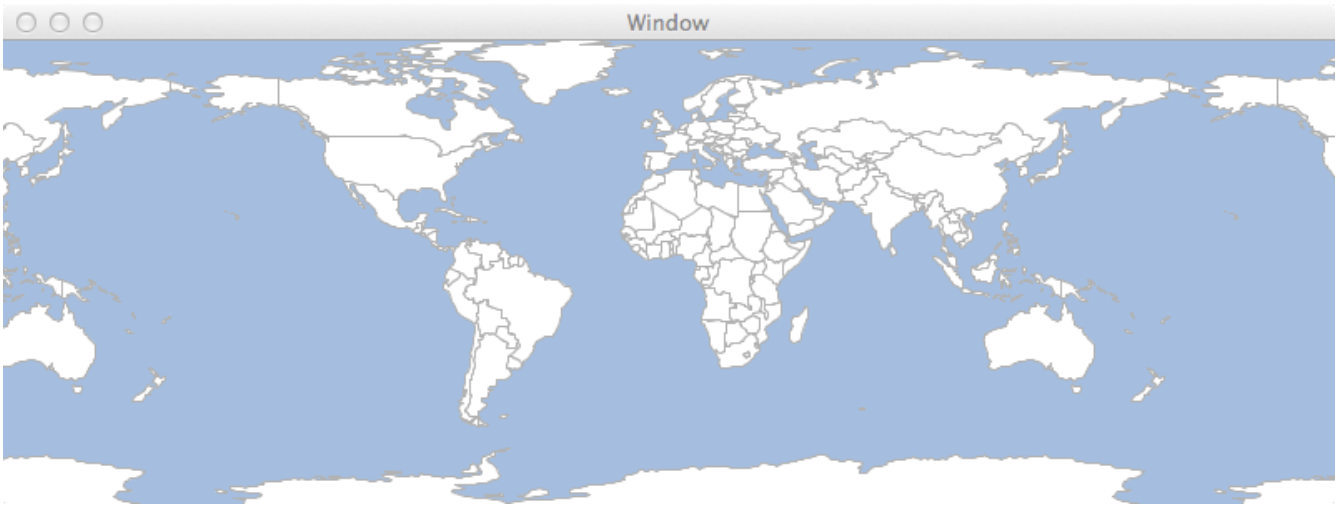
```
Displayer displayer = Displayers.find("window")
println displayer.class.simpleName
```

```
Window
```

Window

Display a Map in a simple GUI

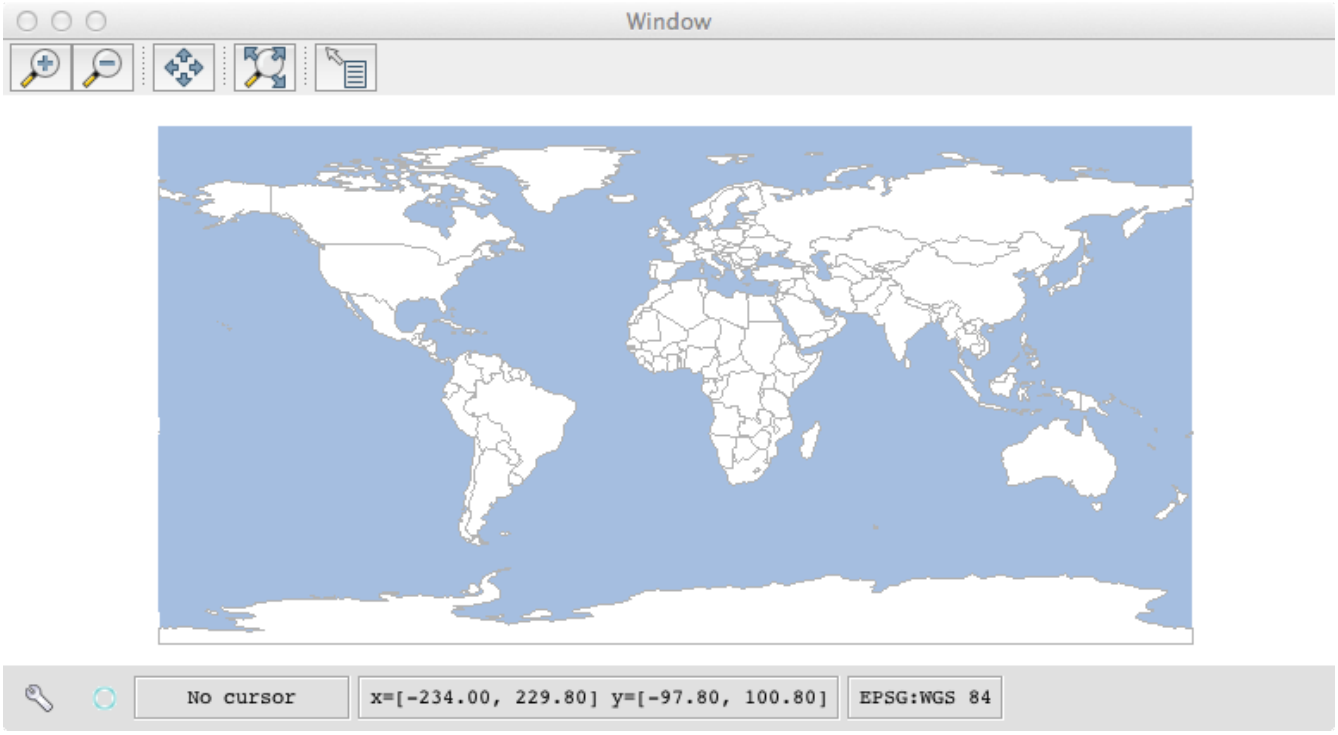
```
Workspace workspace = new GeoPackage('src/main/resources/data.gpkg')
Layer countries = workspace.get("countries")
countries.style = new Fill("#ffffff") + new Stroke("#b2b2b2", 0.5)
Layer ocean = workspace.get("ocean")
ocean.style = new Fill("#a5bfdd")
Map map = new Map(
        width: 800,
        height: 300,
        layers: [ocean, countries]
)
Window window = new Window()
window.display(map)
```



MapWindow

Display a Map in a interactive GUI

```
Workspace workspace = new GeoPackage('src/main/resources/data.gpkg')
Layer countries = workspace.get("countries")
countries.style = new Fill("#ffffff") + new Stroke("#b2b2b2", 0.5)
Layer ocean = workspace.get("ocean")
ocean.style = new Fill("#a5bfdd")
Map map = new Map(
        width: 800,
        height: 300,
        layers: [ocean, countries]
)
MapWindow window = new MapWindow()
window.display(map)
```



Drawing

The Draw class is an easy way to quickly render a Geometry, a List of Geometries, a Feature, or a Layer to an Image, a File, an OutputStream, or a GUI.

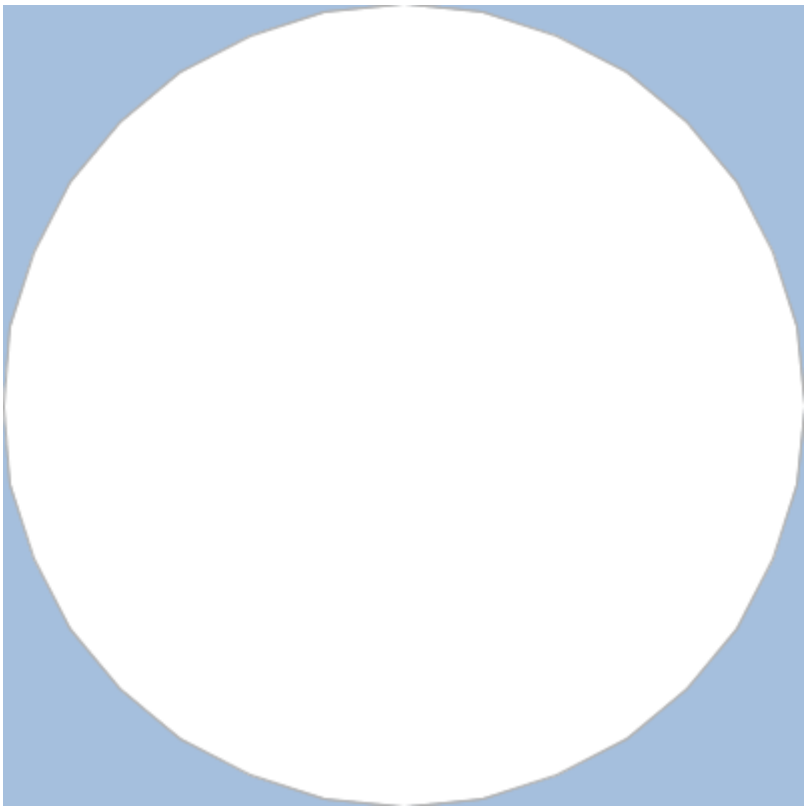
Drawing to a File or GUI

All of the draw methods take a single required parameter but can also take the following optional parameters:

- style = A Style
- bounds = The Bounds
- size = The size of the canvas ([400,350])
- out = The OutputStream, File, or File name. If null (which is the default) a GUI will be opened.
- format = The format ("jpeg", "png", "gif")
- proj = The Projection

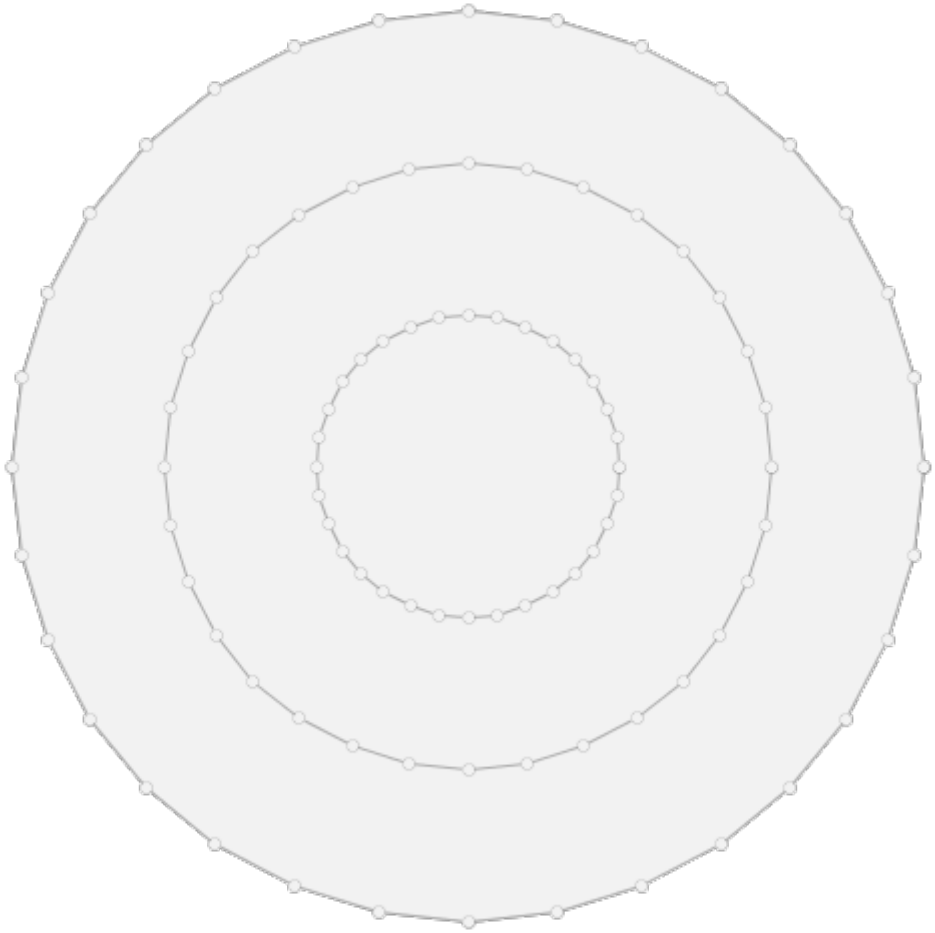
Draw a Geometry to a File

```
File file = new File("geometry.png")
Geometry geometry = new Point(-122.376, 47.587).buffer(0.5)
Draw.draw(geometry,
        style: new Fill("#ffffff") + new Stroke("#b2b2b2", 0.5),
        bounds: new Bounds(-122.876,47.087,-121.876,48.087),
        size: [400,400],
        format: "png",
        proj: "EPSG:4326",
        backgroundColor: "#a5bfdd",
        out: file
)
```



Draw a List of Geometries to an OuputStream

```
Point point = new Point(-122.376, 47.587)
List geometries = [1.5, 1.0, 0.5].collect { double distance ->
    point.buffer(distance)
}
File file = new File("geometries.png")
OutputStream outputStream = new FileOutputStream(file)
Draw.draw(geometries, out: outputStream, format: "png")
outputStream.flush()
outputStream.close()
```

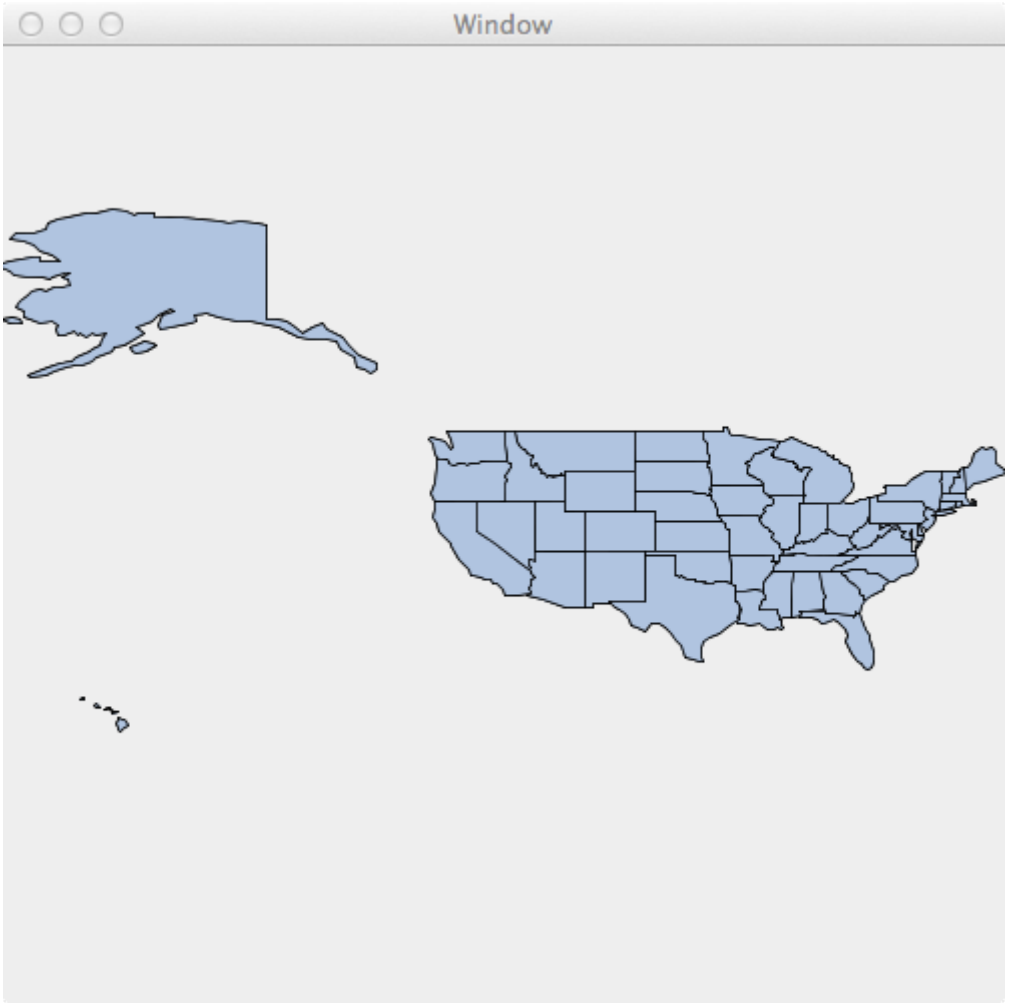


Draw a Feature to a file name

```
Workspace workspace = new GeoPackage('src/main/resources/data.gpkg')
Layer layer = workspace.get("states")
Feature feature = layer.first(filter: "NAME_1='Washington'")
Draw.draw(feature, bounds: feature.bounds, out: "feature.png")
```

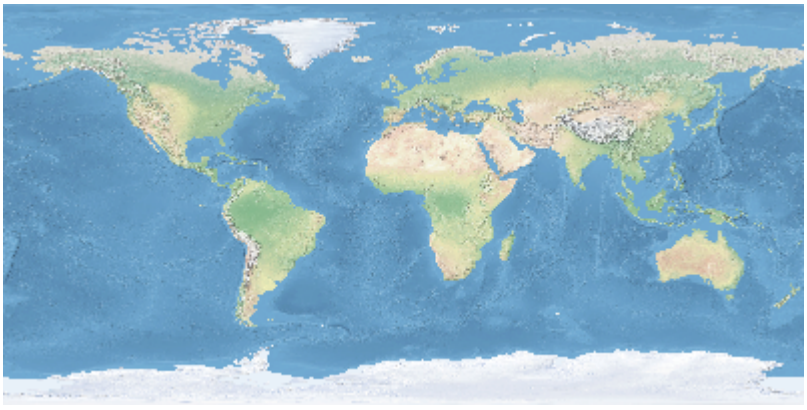
Draw a Layer to a GUI

```
Workspace workspace = new GeoPackage('src/main/resources/data.gpkg')
Layer layer = workspace.get("states")
layer.style = new Fill("LightSteelBlue") + new Stroke("LightSlateGrey", 0.25)
Draw.draw(layer, bounds: layer.bounds)
```



Draw a Raster to a File

```
File file = new File("earth.png")
Raster raster = new geoscript.layer.GeoTIFF(new File('src/main/resources/earth.tif'
)).read()
Draw.draw(raster, bounds: raster.bounds, size: [400,200], out: file)
```



Drawing to an Image

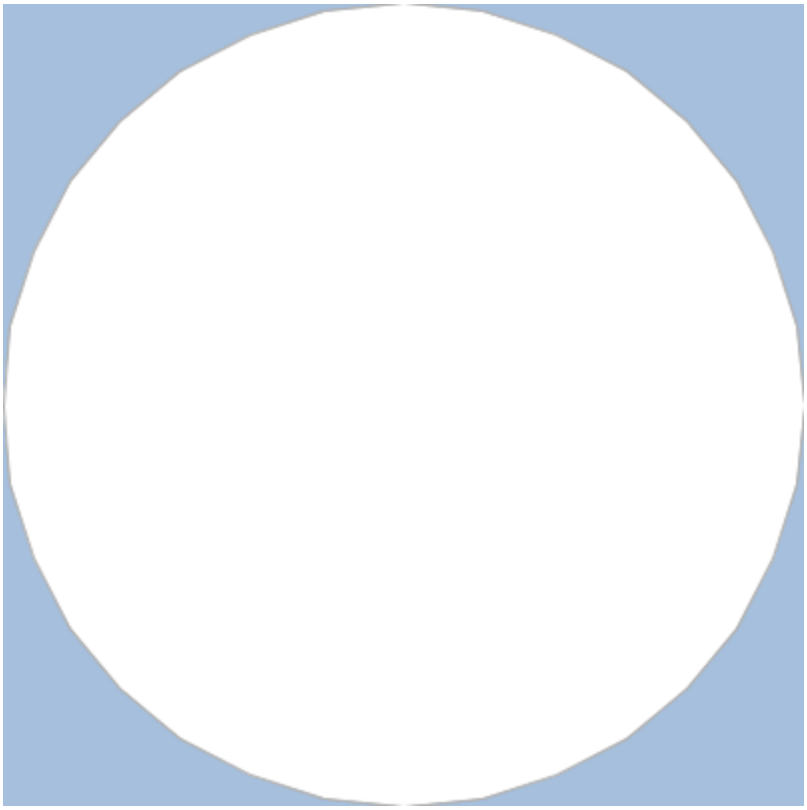
All of the drawToImage methods take a single required parameter but can also take the following optional parameters:

- style = A Style

- bounds = The Bounds
- size = The size of the canvas ([400,350])
- imageType = The format ("jpeg", "png", "gif")
- proj = The Projection

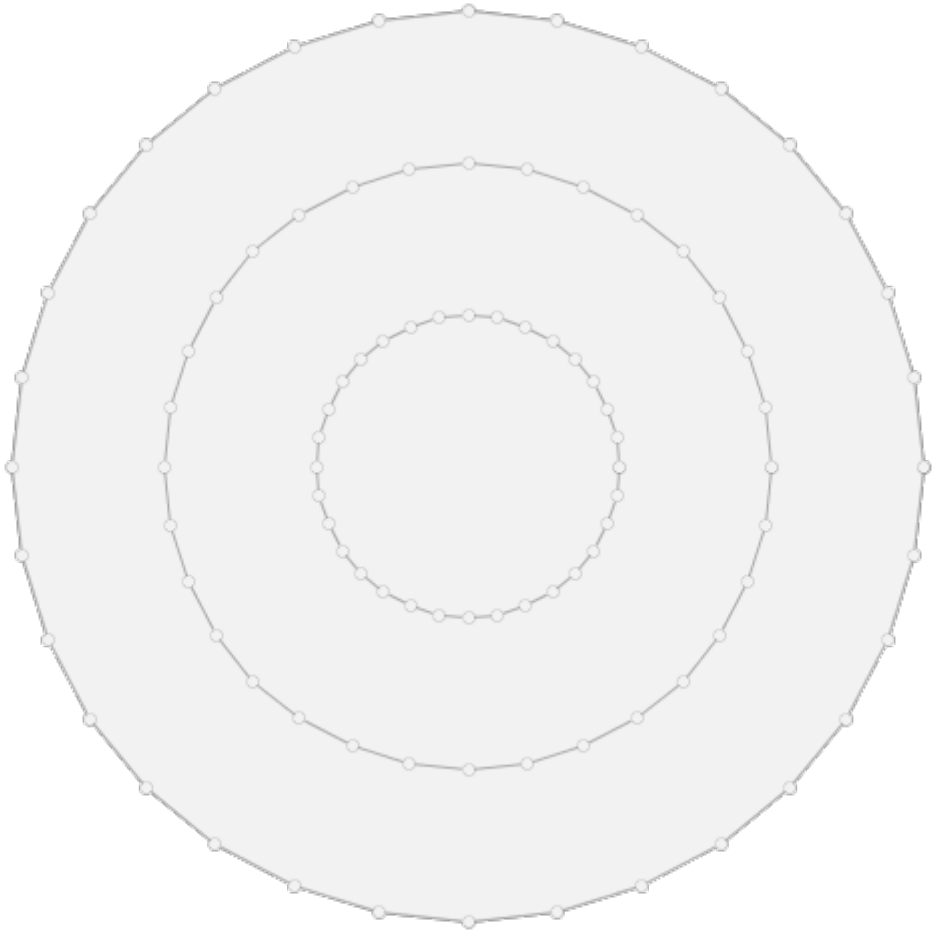
Draw a Geometry to an Image

```
Geometry geometry = new Point(-122.376, 47.587).buffer(0.5)
BufferedImage image = Draw.drawToImage(geometry,
        style: new Fill("#ffffff") + new Stroke("#b2b2b2", 0.5),
        bounds: new Bounds(-122.876,47.087,-121.876,48.087),
        size: [400,400],
        imageType: "png",
        proj: "EPSG:4326",
        backgroundColor: "#a5bfdd"
)
```



Draw a List of Geometries to an Image

```
Point point = new Point(-122.376, 47.587)
List geometries = [1.5, 1.0, 0.5].collect { double distance ->
    point.buffer(distance)
}
BufferedImage image = Draw.drawToImage(geometries)
```

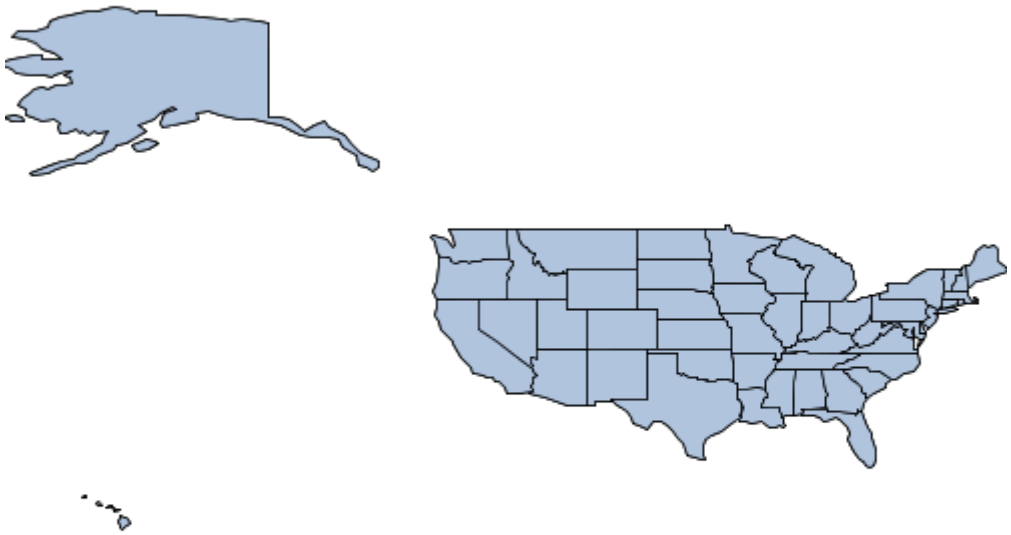


Draw a Feature to an Image

```
Workspace workspace = new GeoPackage('src/main/resources/data.gpkg')
Layer layer = workspace.get("states")
Feature feature = layer.first(filter: "NAME_1='Washington'")
BufferedImage image = Draw.drawToImage(feature, bounds: feature.bounds)
```

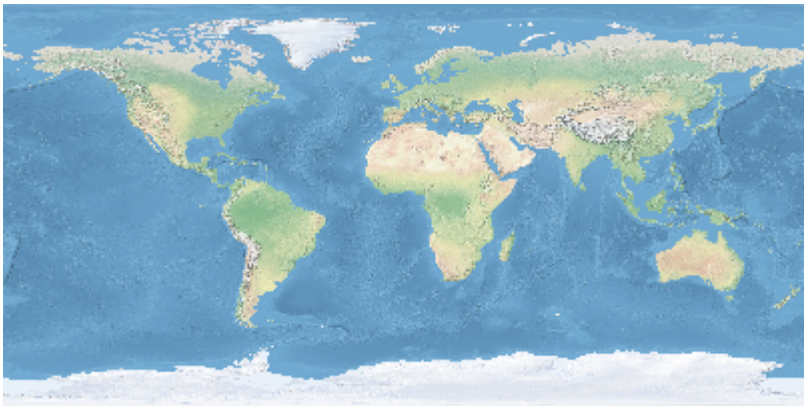
Draw a Layer to an Image

```
Workspace workspace = new GeoPackage('src/main/resources/data.gpkg')
Layer layer = workspace.get("states")
layer.style = new Fill("LightSteelBlue") + new Stroke("LightSlateGrey", 0.25)
BufferedImage image = Draw.drawToImage(layer, bounds: layer.bounds)
```



Draw a Raster to an Image

```
Raster raster = new geoscript.layer.GeoTIFF(new File('src/main/resources/earth.tif'
)).read()
BufferedImage image = Draw.drawToImage(raster, bounds: raster.bounds, size: [400,200])
```



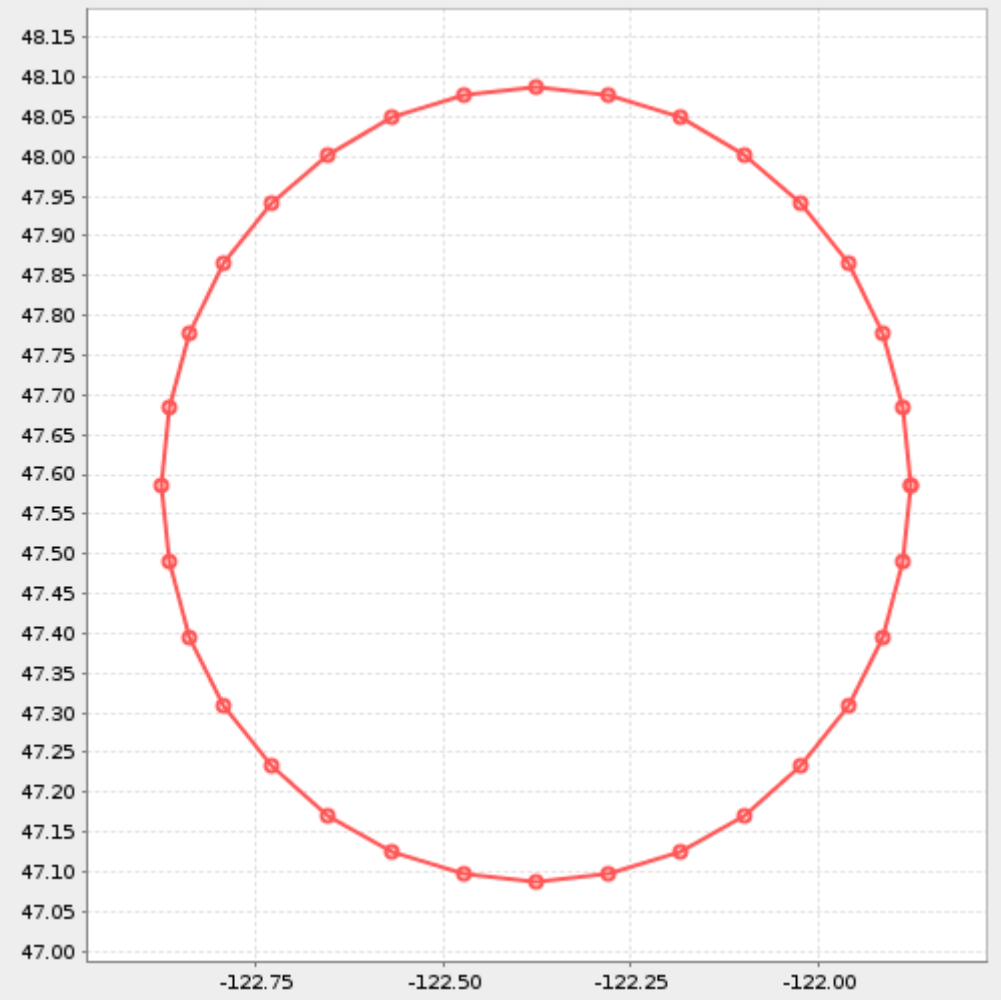
Plotting

Plotting to a File or GUI

The Plot module can plot a Geometry, a list of Geometries, a Feature, or a Layer to a File, a File name, an OutputStream, or a simple GUI.

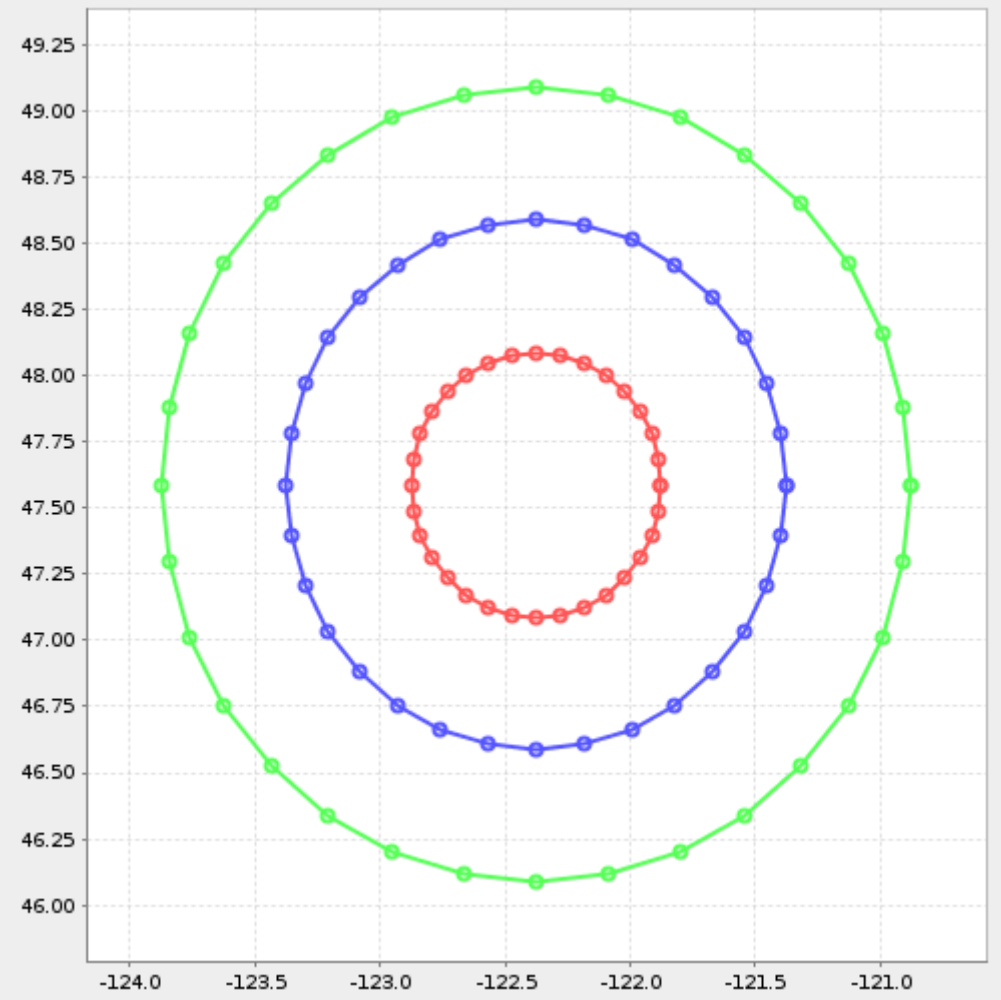
Plot a Geometry to a File

```
File file = new File("geometry.png")
Geometry geometry = new Point(-122.376, 47.587).buffer(0.5)
Plot.plot(geometry, out: file)
```



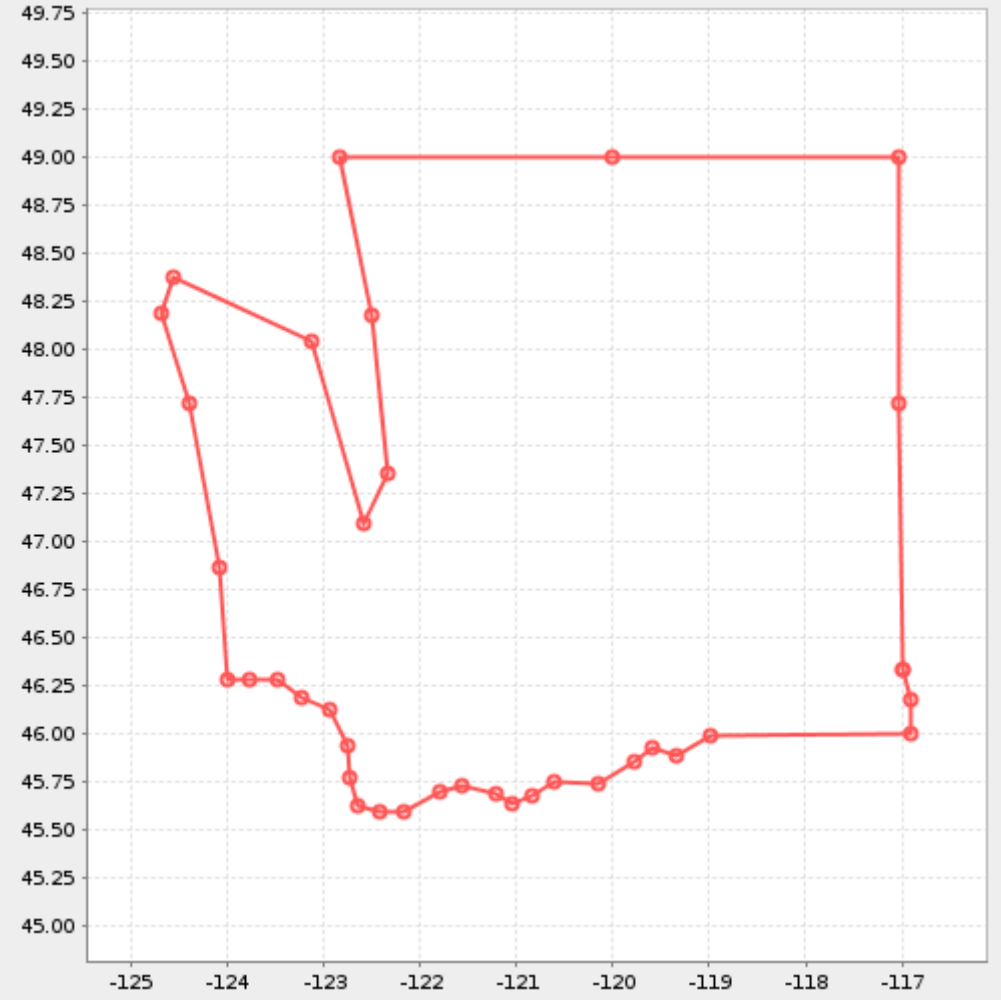
Plot a List of Geometries to an OutputStream

```
Point point = new Point(-122.376, 47.587)
List geometries = [1.5, 1.0, 0.5].collect { double distance ->
    point.buffer(distance)
}
File file = new File("geometries.png")
OutputStream outputStream = new FileOutputStream(file)
Plot.plot(geometries, out: outputStream)
outputStream.flush()
outputStream.close()
```



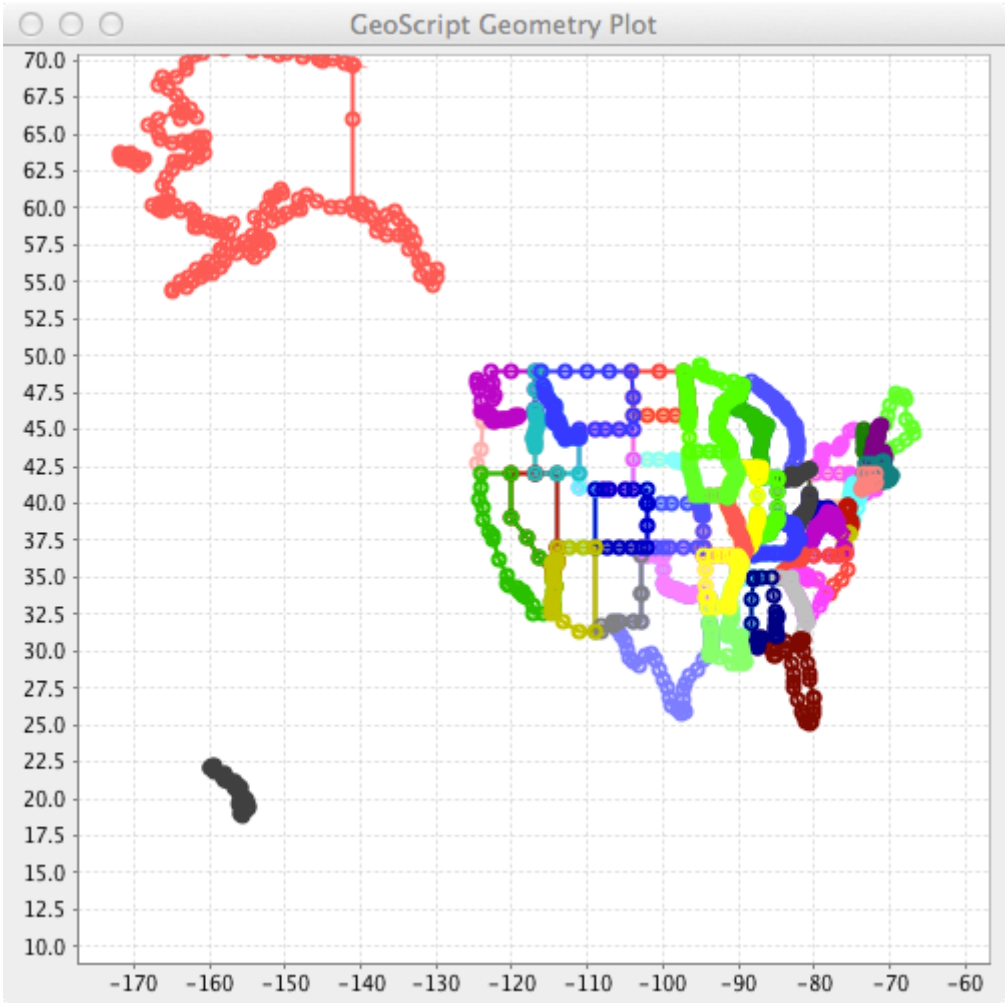
Plot a Feature to a File name

```
Workspace workspace = new GeoPackage('src/main/resources/data.gpkg')
Layer layer = workspace.get("states")
Feature feature = layer.first(filter: "NAME_1='Washington'")
Plot.plot(feature, out: "feature.png")
```



Plot a Layer to a GUI

```
Workspace workspace = new GeoPackage('src/main/resources/data.gpkg')
Layer layer = workspace.get("states")
Plot.plot(layer)
```

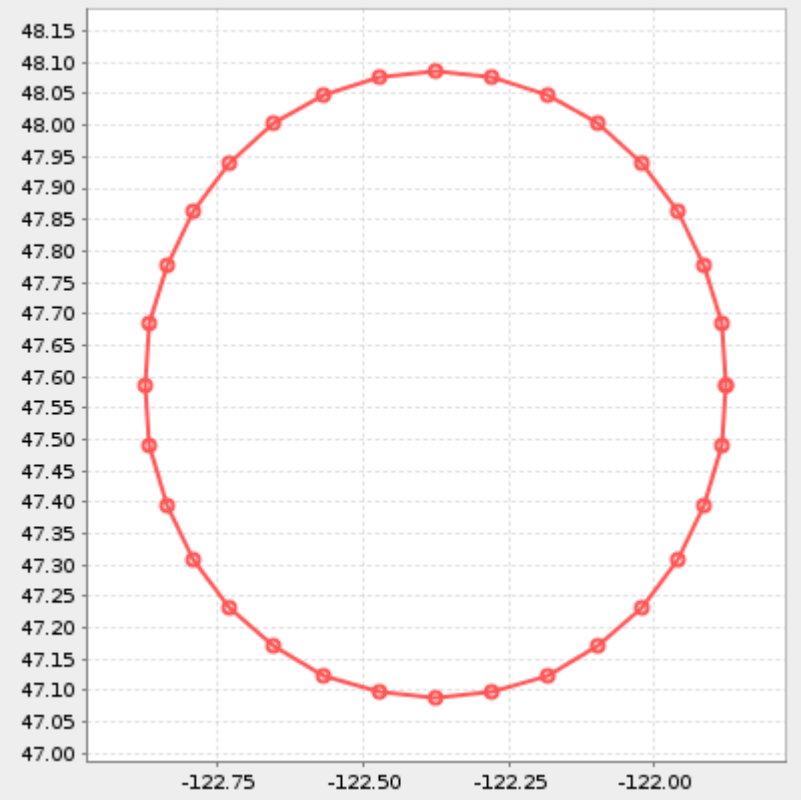


Plotting to an Image

The Plot module can plot a Geometry, a list of Geometries, a Feature, or a Layer to an image.

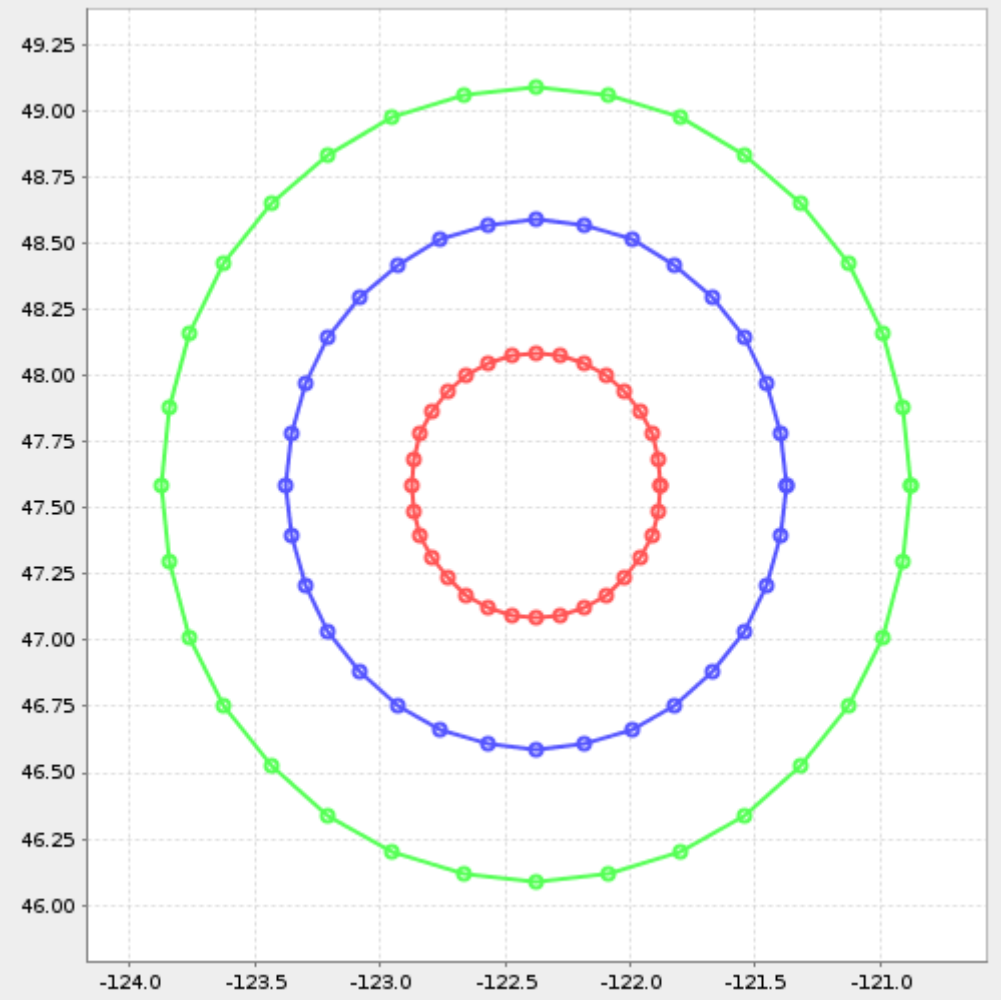
Plot a Geometry to an Image

```
Geometry geometry = new Point(-122.376, 47.587).buffer(0.5)
BufferedImage image = Plot.plotToImage(geometry, size: [400,400],)
```



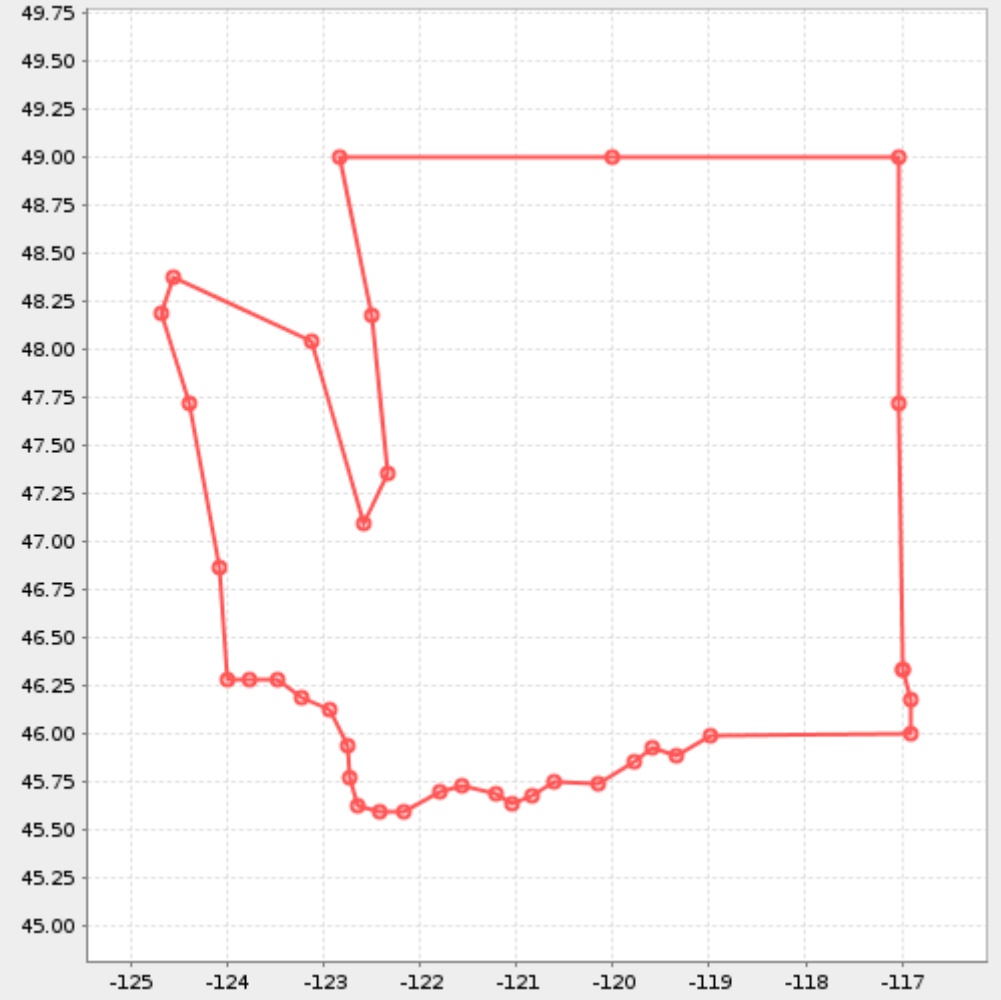
Plot a List of Geometries to an Image

```
Point point = new Point(-122.376, 47.587)
List geometries = [1.5, 1.0, 0.5].collect { double distance ->
    point.buffer(distance)
}
BufferedImage image = Plot.plotToImage(geometries)
```



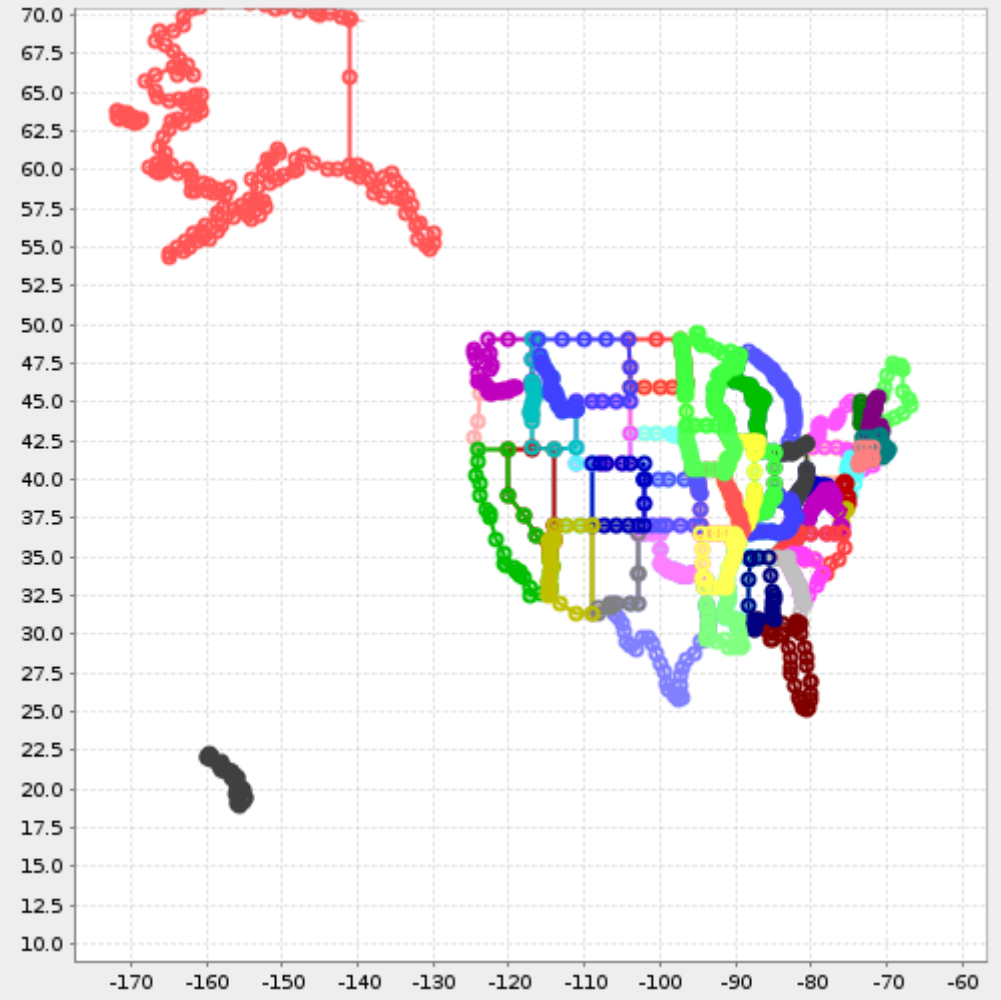
Plot a Feature to an Image

```
Workspace workspace = new GeoPackage('src/main/resources/data.gpkg')
Layer layer = workspace.get("states")
Feature feature = layer.first(filter: "NAME_1='Washington'")
BufferedImage image = Plot.plotToImage(feature, bounds: feature.bounds)
```



Plot a Layer to an Image

```
Workspace workspace = new GeoPackage('src/main/resources/data.gpkg')
Layer layer = workspace.get("states")
BufferedImage image = Plot.plotToImage(layer, bounds: layer.bounds)
```



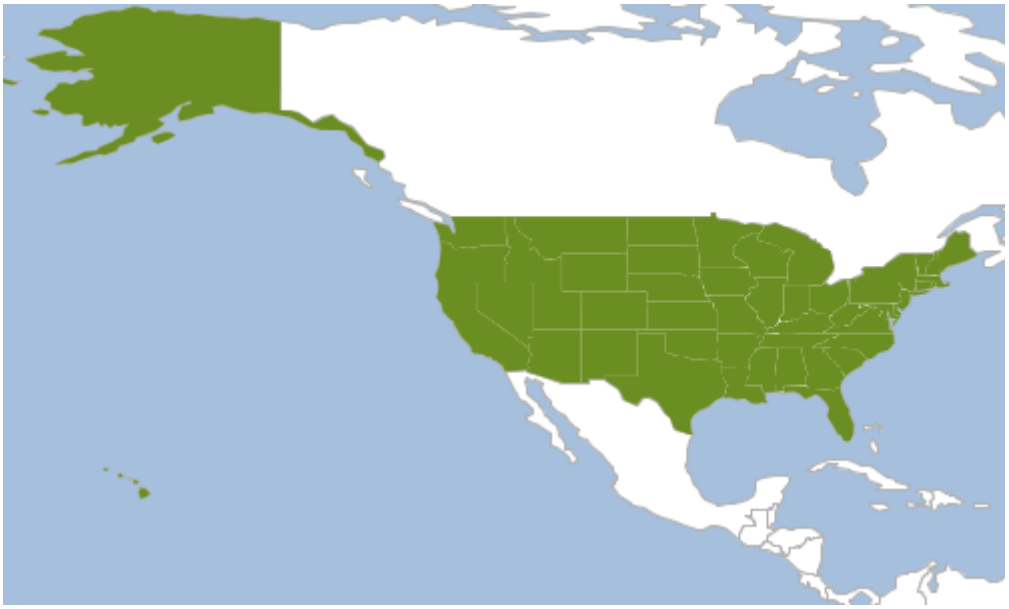
Style Recipes

Styles are found in the **geoscript.style** package.

Styles are made up Symbolizers and Composites. A Symbolizer is a particular style like Stroke or Fill. Symbolizers also have methods for controlling the drawing order (zindex), the min and max scale (range), and filtering (where).

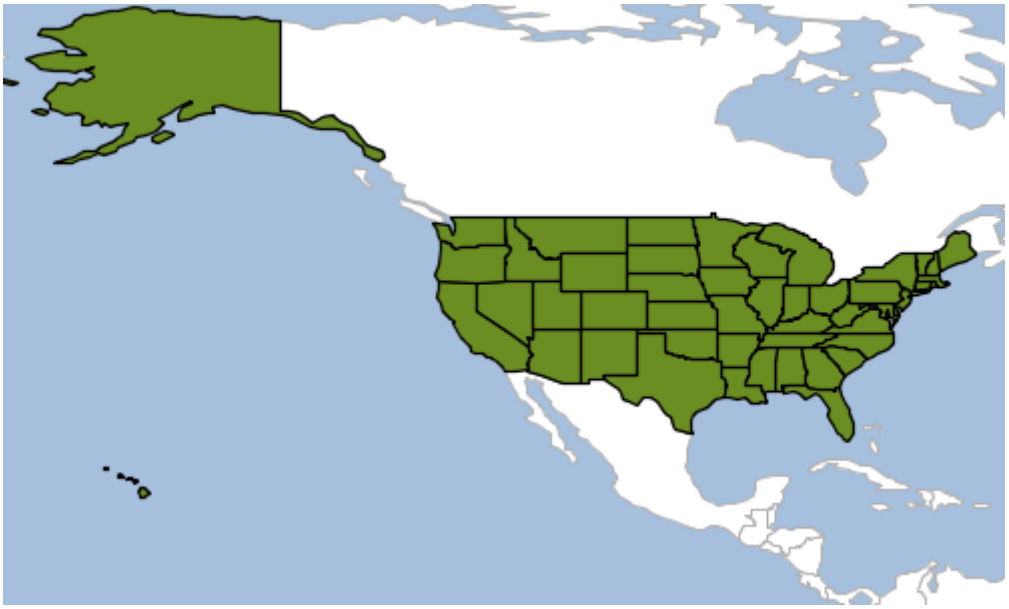
Creating Basic Styles

```
Fill fill = new Fill("#6B8E23")
```



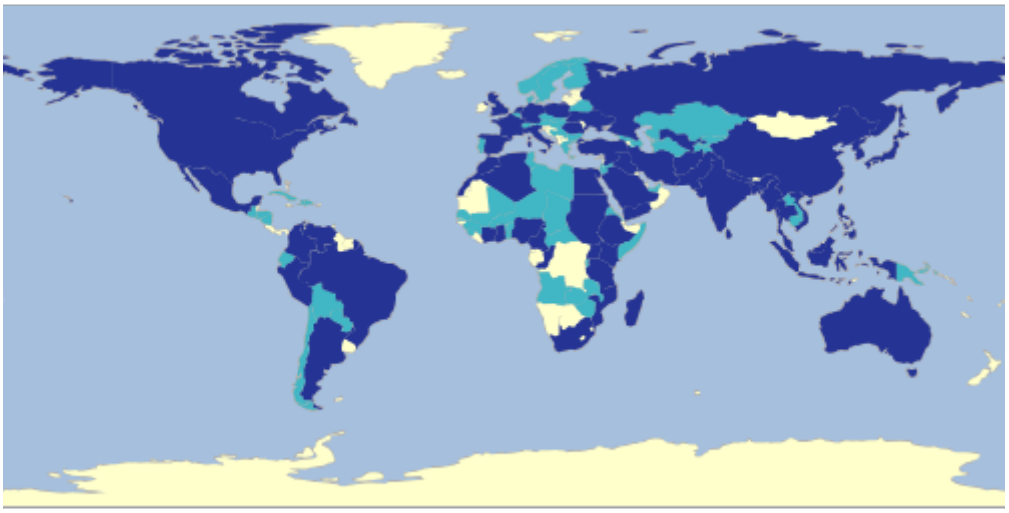
A Composite is simply two or more Symbolizers. So, a Composite would be a combination of a Stroke symbolizer (to style the boundary) and a Fill Symbolizer (to style the interior).

```
Composite composite = new Fill("#6B8E23") + new Stroke("black", 0.75)
```



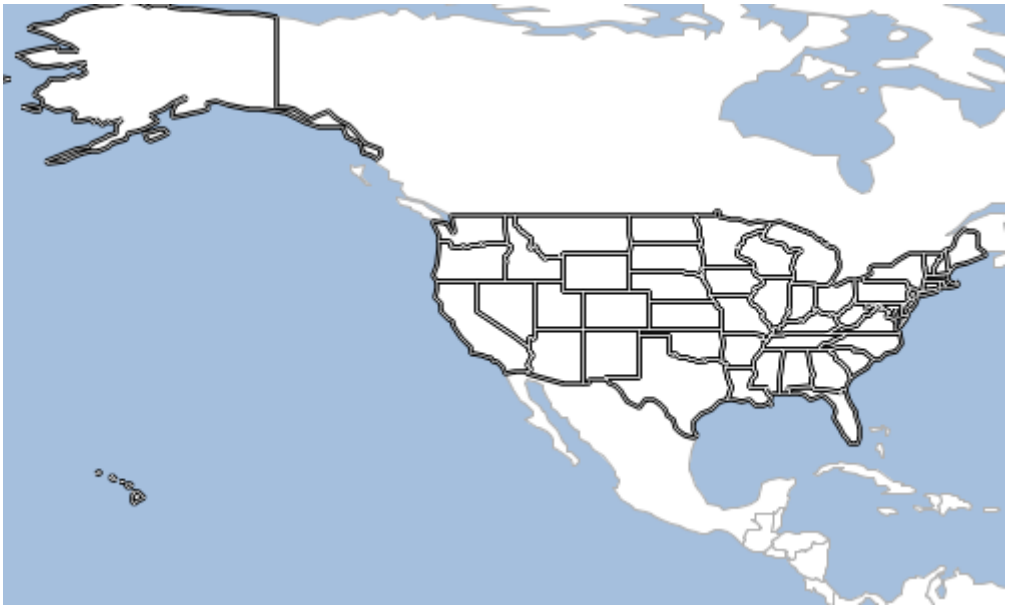
A Symbolizer can use the where method to restrict which features are styled.

```
Symbolizer symbolizer = new Fill("#ffffcc").where("PEOPLE < 4504128.33") +
        new Fill("#41b6c4").where("PEOPLE BETWEEN 4504128.33 AND 16639804.33") +
        new Fill("#253494").where("PEOPLE > 16639804.33")
```



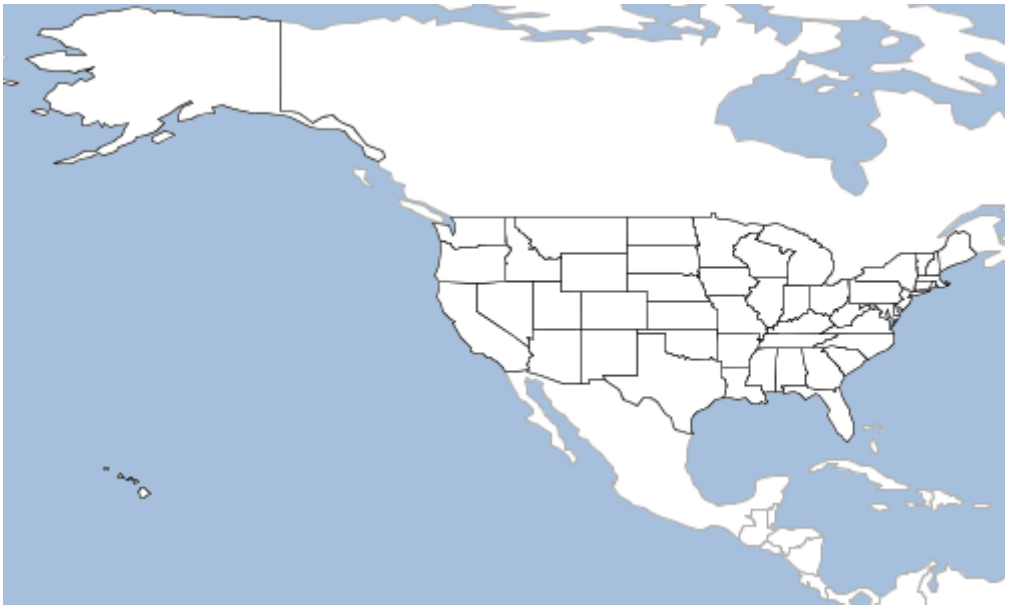
The zindex method is used to order Symbolizers on top of each other. In this recipe we use it to create line casings.

```
Symbolizer symbolizer = new Stroke("black", 2.0).zindex(0) + new Stroke("white", 0.1
).zindex(1)
```



The scale method is used to create Symbolizers that are dependent on map scale.

```
Symbolizer symbolizer = (new Fill("white") + new Stroke("black", 0.1)) + new Label
("NAME_1")
        .point(anchor: [0.5,0.5])
        .polygonAlign("mbr")
        .range(max: 16000000)
```



Creating Strokes

Create a Stroke Symbolizer with a Color

```
Stroke stroke = new Stroke("#1E90FF")
```

Create a Stroke Symbolizer with a Color and Width

```
Stroke stroke = new Stroke("#1E90FF", 0.5)
```

Create a Stroke Symbolizer with casing

```
Symbolizer stroke = new Stroke(color: "#333333", width: 5, cap: "round").zindex(0) +
        new Stroke(color: "#6699FF", width: 3, cap: "round").zindex(1)
```

Create a Stroke Symbolizer with Dashes

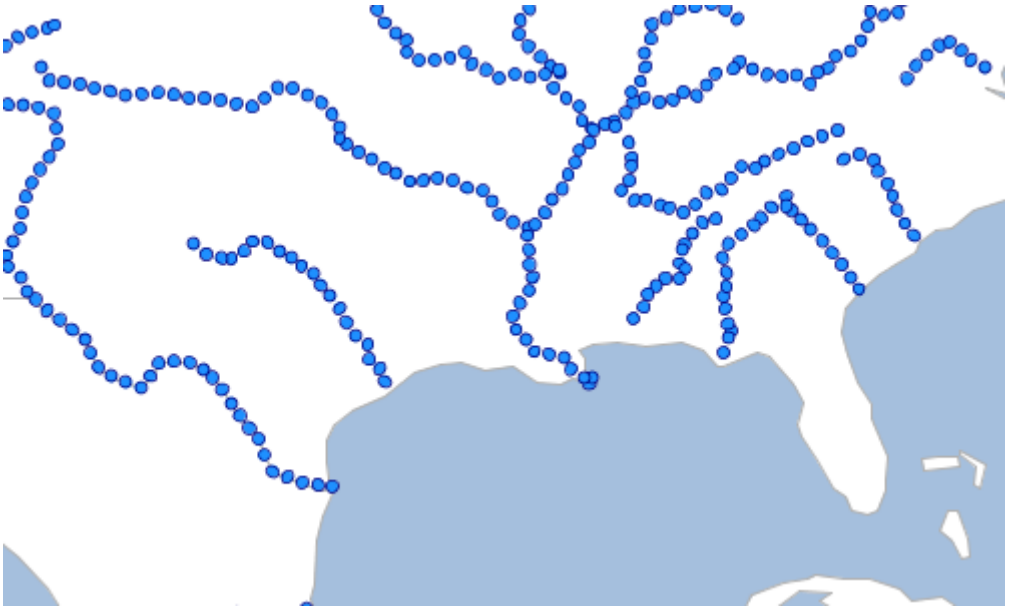
```
Stroke stroke = new Stroke("#1E90FF", 0.75, [5,5], "round", "bevel")
```

Create a Stroke Symbolizer with railroad Hatching

```
Symbolizer stroke = new Stroke("#1E90FF", 1) + new Hatch("vertline", new Stroke
("#1E90FF", 0.5), 6).zindex(1)
```

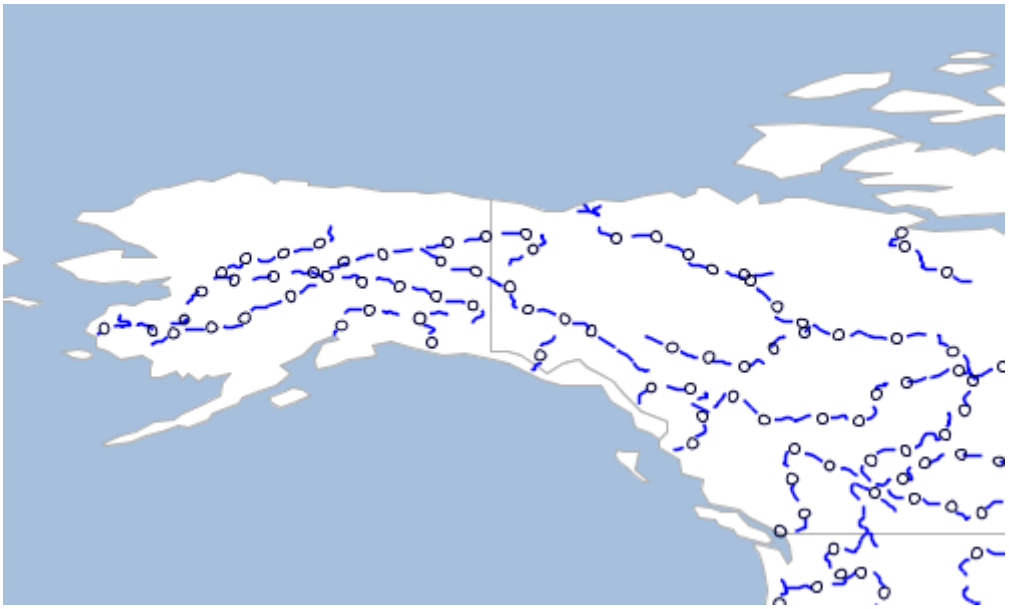
Create a Stroke Symbolizer with spaced Shape symbols

```
Symbolizer stroke = new Stroke(width: 0, dash: [4, 4]).shape(new Shape("#1E90FF", 6,
"circle").stroke("navy", 0.75))
```



Create a Stroke Symbolizer with alternating spaced Shape symbols

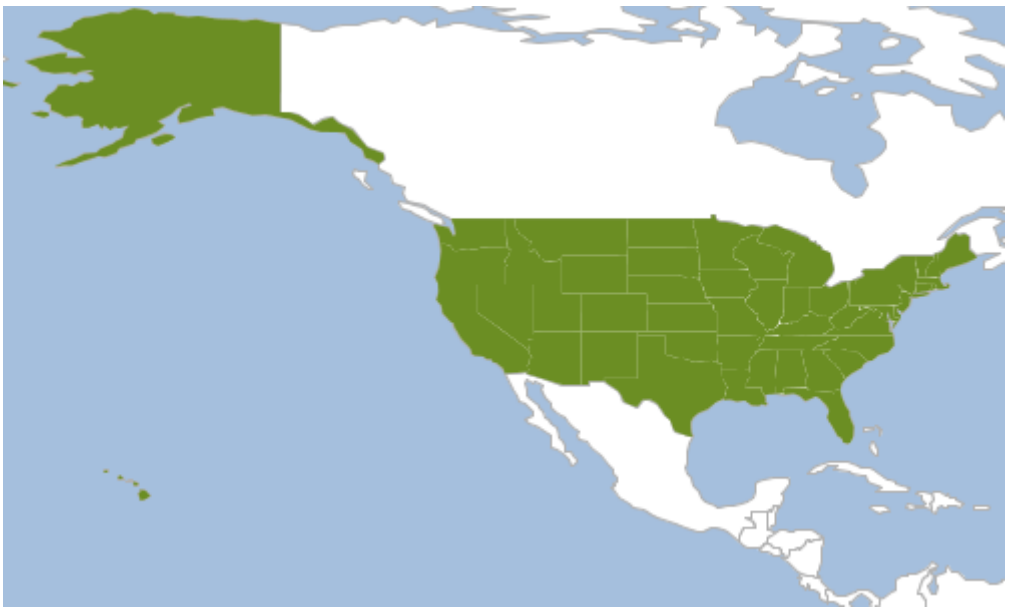
```
Symbolizer stroke = new Stroke("#0000FF", 1, [10,10]).zindex(0) + new Stroke(null, 0,
[[5,15],7.5])
        .shape(new Shape(null, 5, "circle").stroke("#000033",1)).zindex(1)
```



Creating Fills

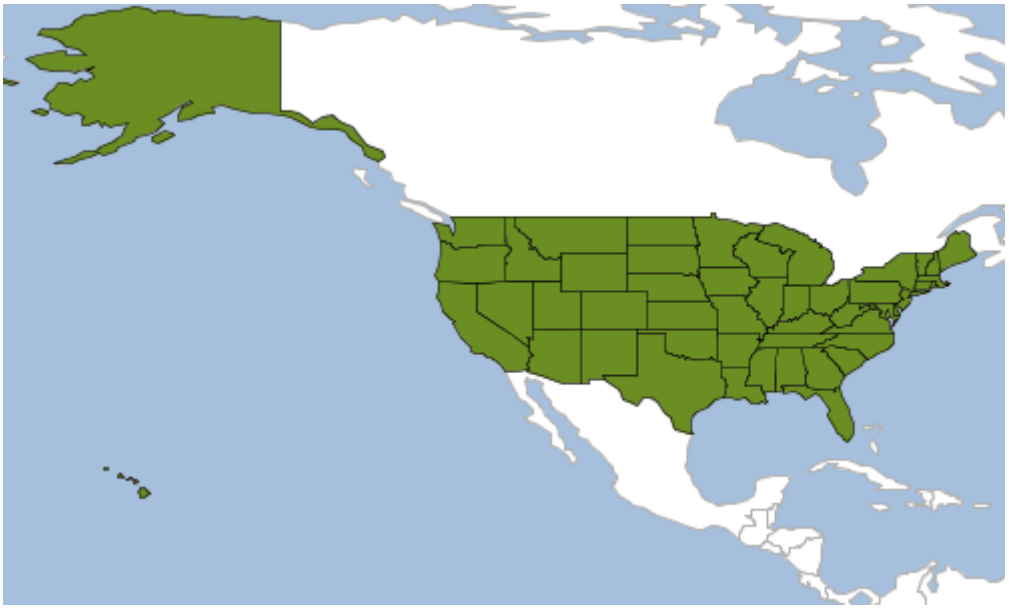
Create a Fill Symbolizer with a Color

```
Fill fill = new Fill("#6B8E23")
```



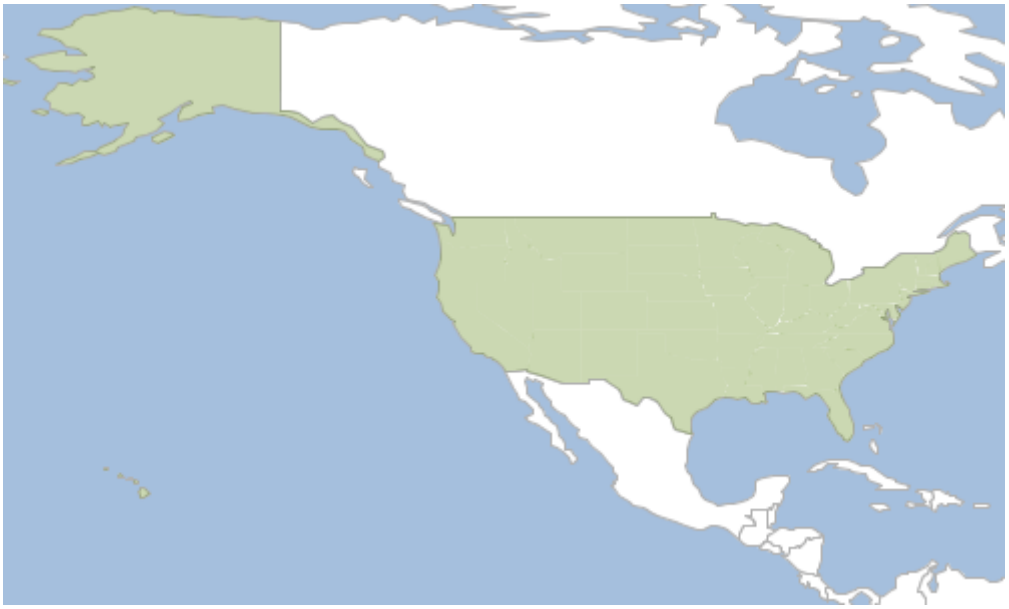
Create a Fill Symbolizer with a Color and a Stroke

```
Symbolizer symbolizer = new Fill("#6B8E23") + new Stroke("black", 0.1)
```



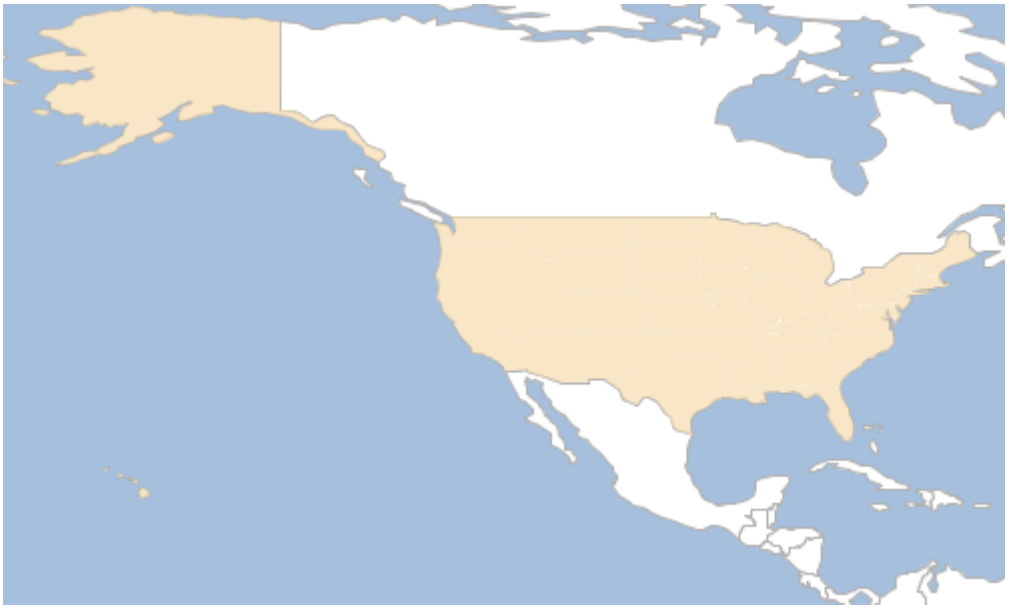
Create a Fill Symbolizer with a Color and Opacity

```
Fill fill = new Fill("#6B8E23", 0.35)
```



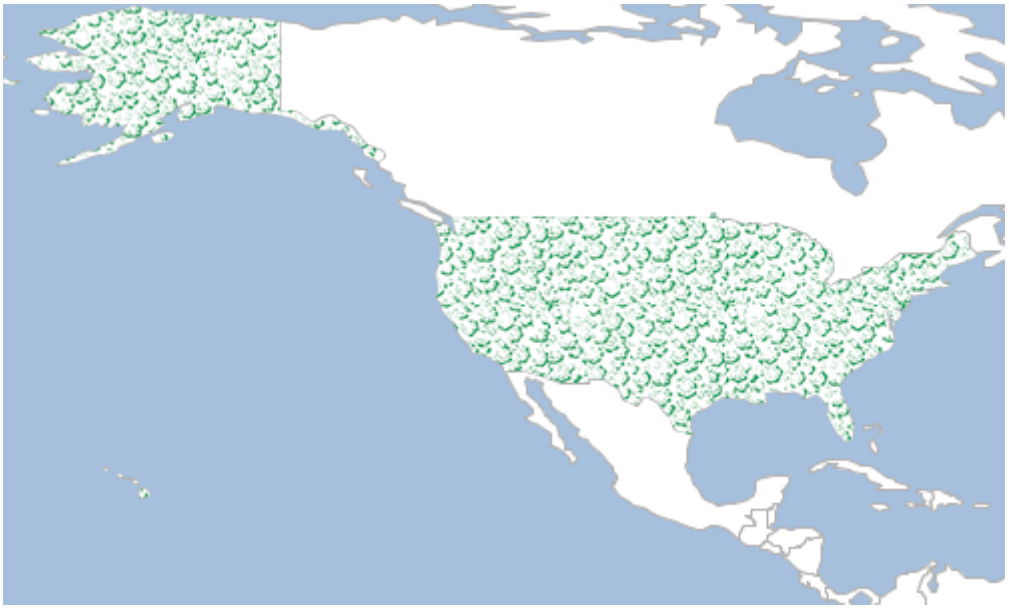
Create a Fill Symbolizer from named parameters

```
Fill fill = new Fill(color: "wheat", opacity: 0.75)
```



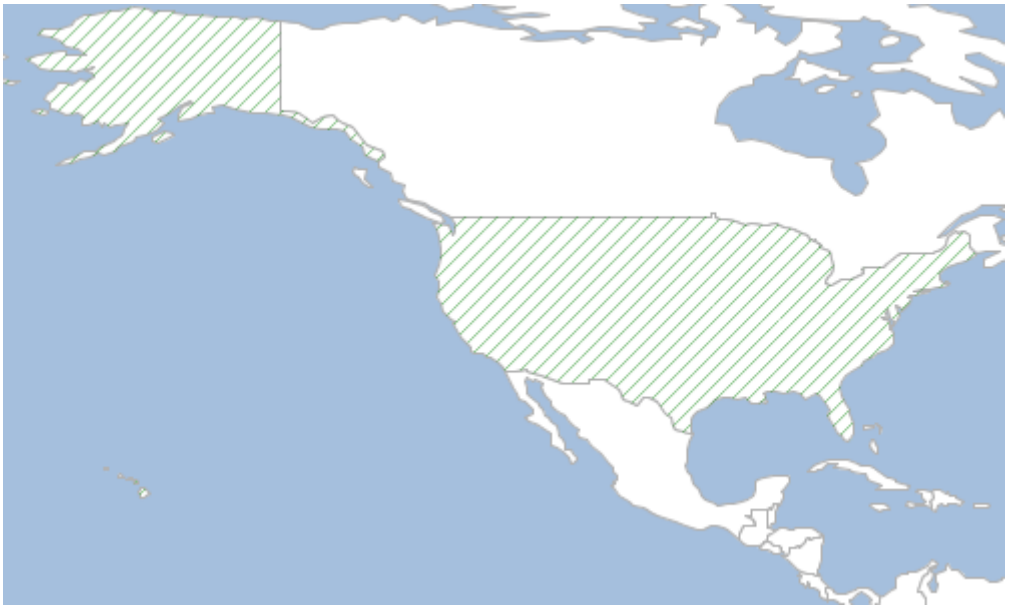
Create a Fill Symbolizer with an Icon

```
Fill fill = new Fill("green").icon('src/main/resources/trees.png', 'image/png')
```



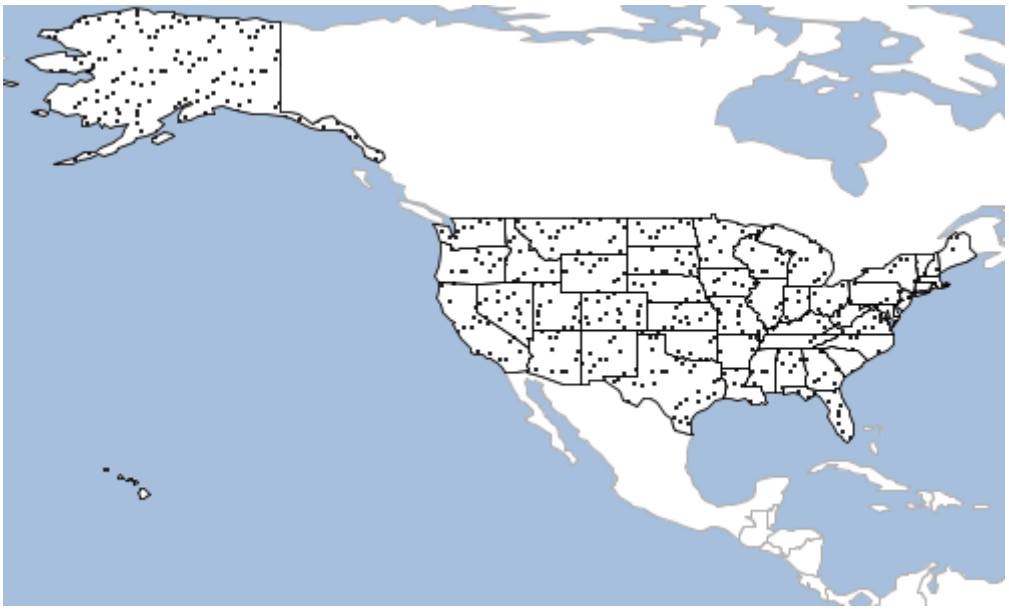
Create a Fill Symbolizer with a Hatch

```
Fill fill = new Fill("green").hatch("slash", new Stroke("green", 0.25), 8)
```



Create a Fill Symbolizer with a random fill

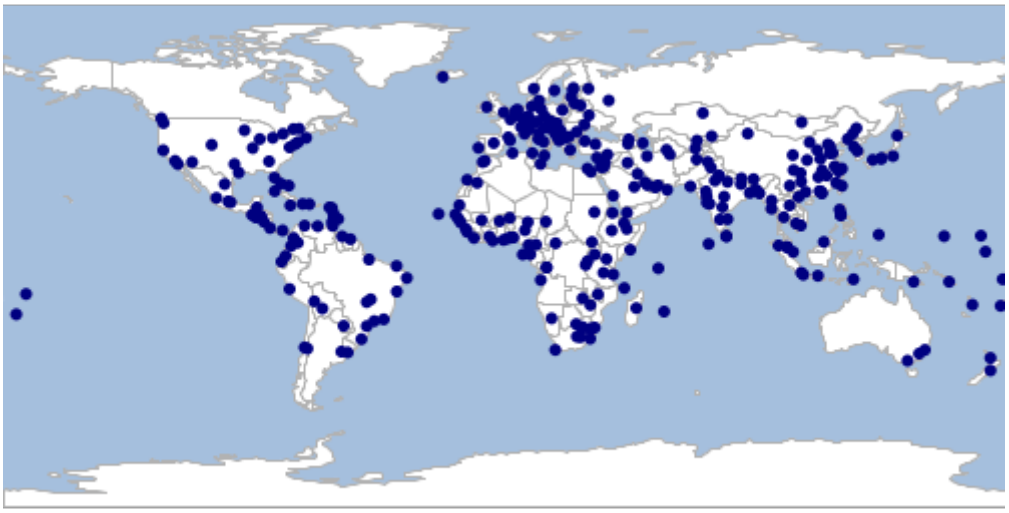
```
Symbolizer symbolizer = new Fill("white").hatch("circle", new Fill("black"), 2).
random(
    random: "free",
    seed: 0,
    symbolCount: 50,
    tileSize: 50,
    rotation: "none"
) + new Stroke("black", 0.25)
```



Creating Shapes

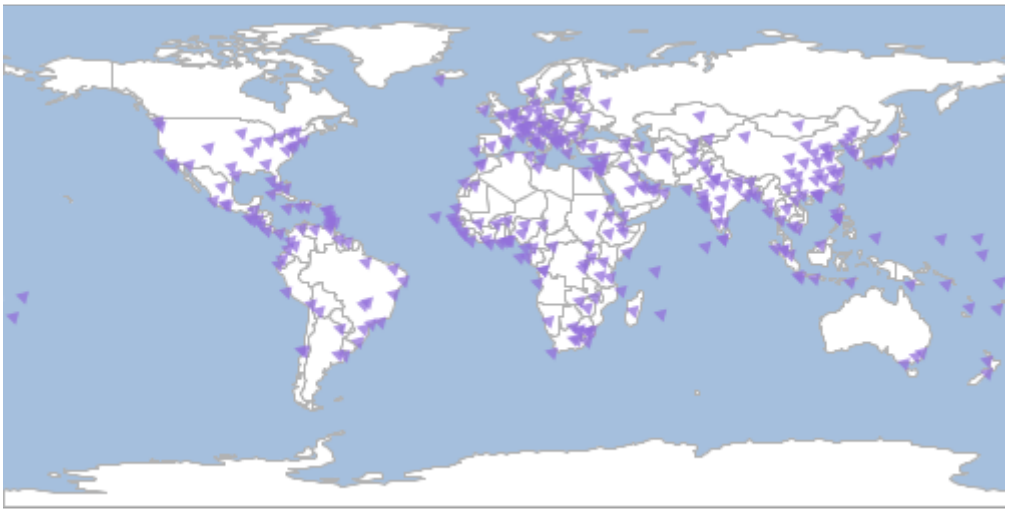
Create a Shape Symbolizer with a Color

```
Shape shape = new Shape("navy")
```



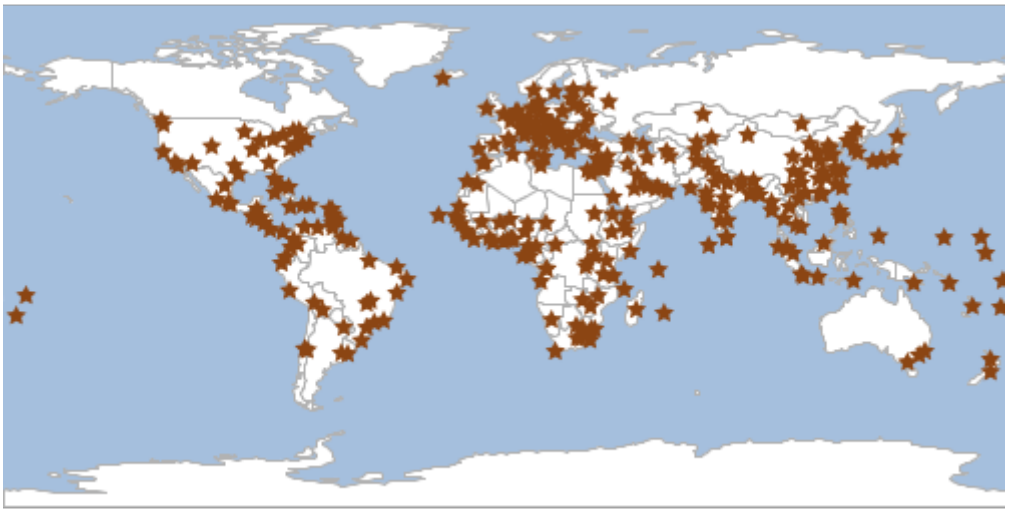
Create a Shape Symbolizer with a color, size, type, opacity and angle

```
Shape shape = new Shape("#9370DB", 8, "triangle", 0.75, 45)
```



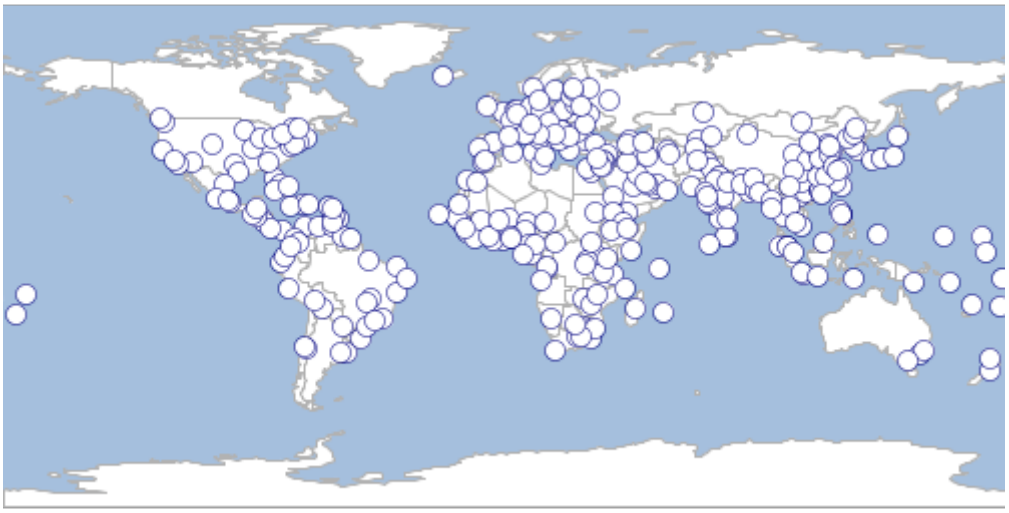
Create a Shape Symbolizer with named parameters

```
Shape shape = new Shape(color: "#8B4513", size: 10, type: "star", opacity: 1.0,
rotation: 0)
```



Create a Shape Symbolizer with Stroke outline

```
Symbolizer symbolizer = new Shape("white", 10).stroke("navy", 0.5)
```



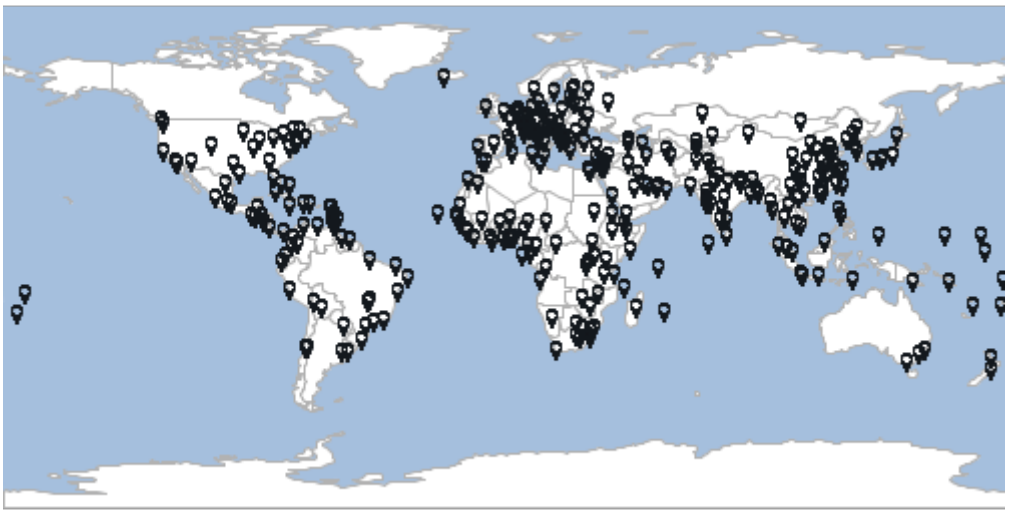
Creating Icons

Create an Icon Symbolizer

```
Symbolizer symbolizer = new Icon("src/main/resources/place.png", "image/png", 12)
```

Create an Icon Symbolizer

```
Symbolizer symbolizer = new Icon(url: "src/main/resources/place.png", format:
"image/png", size: 10)
```



Creating Labels

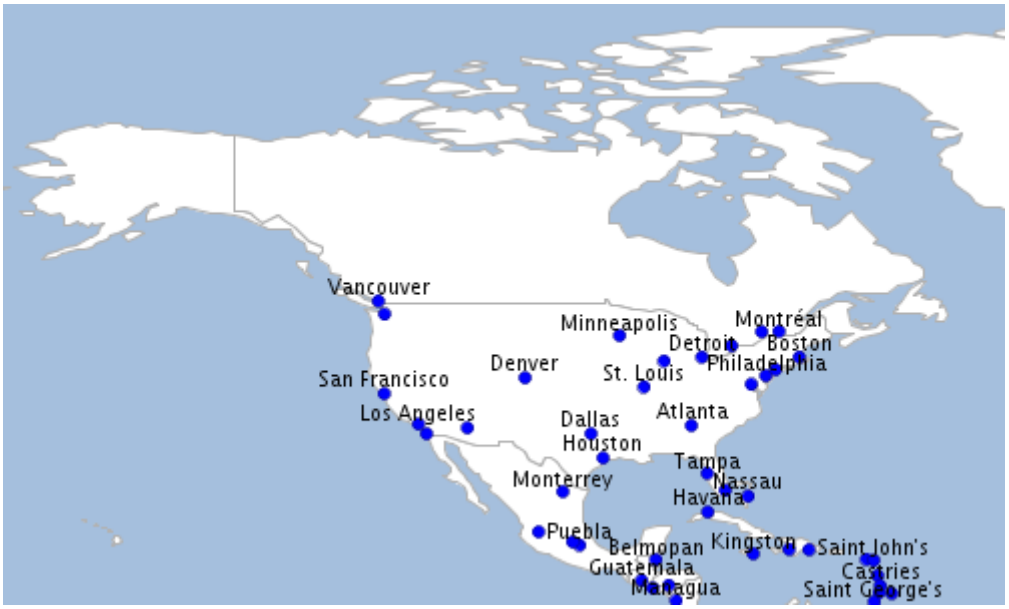
Create a Label for a Point Layer

```
Symbolizer symbolizer = new Shape("blue", 6).stroke("navy", 0.5) + new Label("NAME"
).point(
        [0.5,0.5],  ①
        [0, 5.0],   ②
        0           ③
)
```

① anchor

② displacement

③ rotation



Create a Label for a Point Layer with a Font

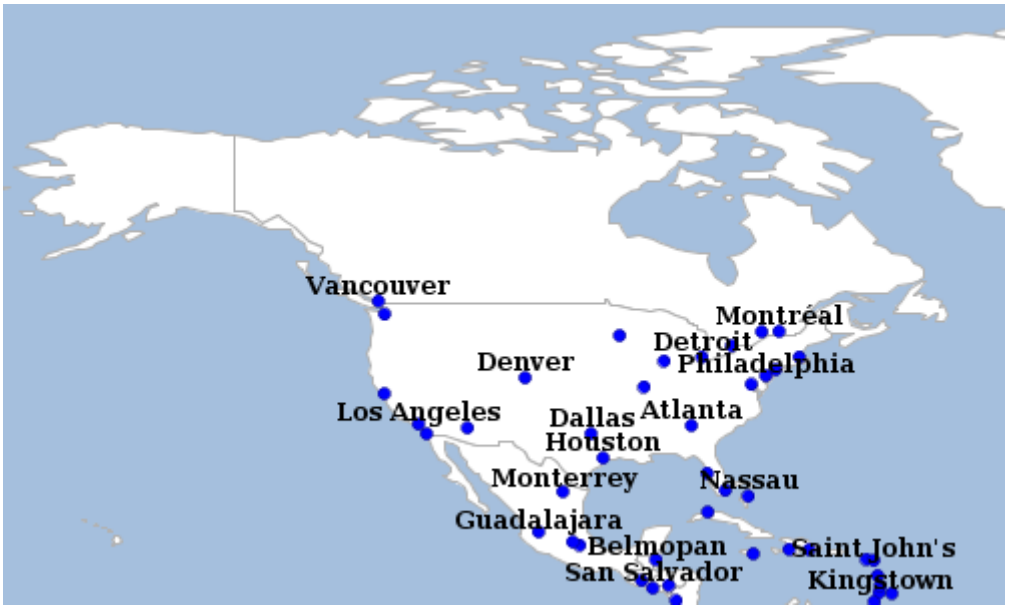
```
Symbolizer symbolizer = new Shape("blue", 6).stroke("navy", 0.5) + new Label("NAME"
).point(
        [0.5,0.5],
        [0, 5.0],
        0
) + new Font(
        "normal",    ①
        "bold",      ②
        12,          ③
        "Arial"      ④
)
```

① style (normal, italic, oblique)

② weight (normal, bold)

③ size (8,12,16,ect..)

④ family (serif, arial, verdana)



Create a Label for a Point Layer with Halo

```
Symbolizer symbolizer = new Shape("blue", 6).stroke("navy", 0.5) + new Label("NAME"
).point(
        [0.5,0.5],
        [0, 5.0],
        0
).fill(new Fill("white")) + new Halo(new Fill("navy"), 2.5)
```

Create a Label for a Polygon Layer

```
Symbolizer symbolizer = new Fill("white") + new Stroke("black", 0.1) + new Label
("NAME_1")
    .point(anchor: [0.5,0.5])
    .polygonAlign("mbr")
```

Create a Label for a Line Layer

```
Symbolizer symbolizer = new Stroke("blue", 0.75)  + new Label("name")
        .fill(new Fill("navy"))
        .linear(follow: true, offset: 50, displacement: 200, repeat: 150
).maxDisplacement(400).maxAngleDelta(90)
        .halo(new Fill("white"), 2.5)
        .font(new Font(size: 10, weight: "bold"))
```

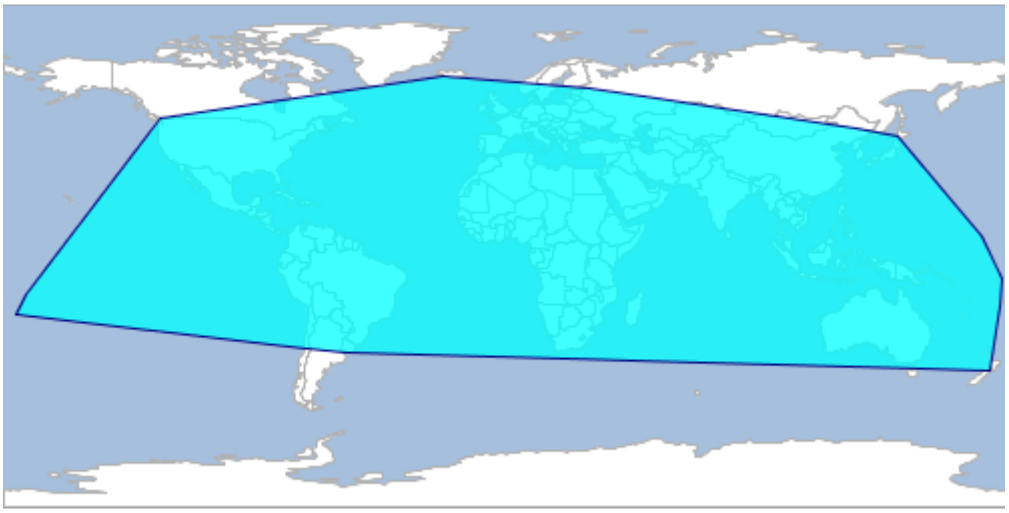
Creating Transforms

Create a normal Transform symbolizer that styles a polygon as a point by calculating it's centroid

```
Workspace workspace = new GeoPackage('src/main/resources/data.gpkg')
Layer countries = workspace.get("countries")
Symbolizer symbolizer = new Transform("centroid(the_geom)") +
        new Shape(color: "red", size: 10, type: "star")
countries.style = symbolizer
```

Create a rendering Transform symbolizer that styles a point layer by calculating the convex hull

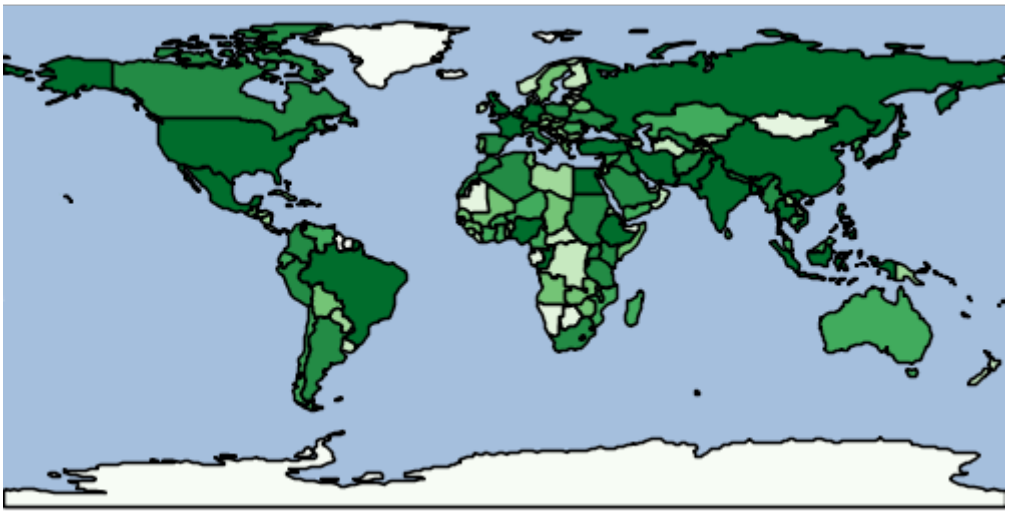
```
Workspace workspace = new GeoPackage('src/main/resources/data.gpkg')
Layer places = workspace.get("places")
Process process = new Process("convexhull",
        "Create a convexhull around the features",
        [features: geoscript.layer.Cursor],
        [result: geoscript.layer.Cursor],
        { inputs ->
            def geoms = new GeometryCollection(inputs.features.collect{ f -> f.geom})
            def output = new Layer()
            output.add([geoms.convexHull])
            [result: output]
        }
)
Function function = new Function(process, new Function("parameter", new Expression
("features")))
Symbolizer symbolizer =  new Transform(function, Transform.RENDERING) + new Fill
("aqua", 0.75) + new Stroke("navy", 0.5)
places.style = symbolizer
```



Creating Gradients

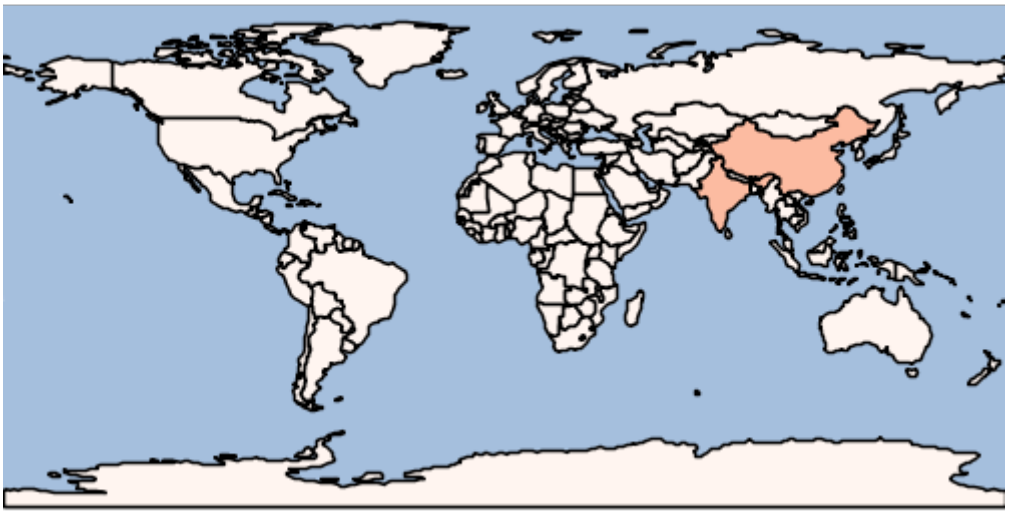
Create a Gradient Symbolizer from a Layer's Field using quantile method

```
Workspace workspace = new GeoPackage('src/main/resources/data.gpkg')
Layer countries = workspace.get("countries")
Gradient gradient = new Gradient(countries, "PEOPLE", "quantile", 8, "Greens")
countries.style = gradient
```



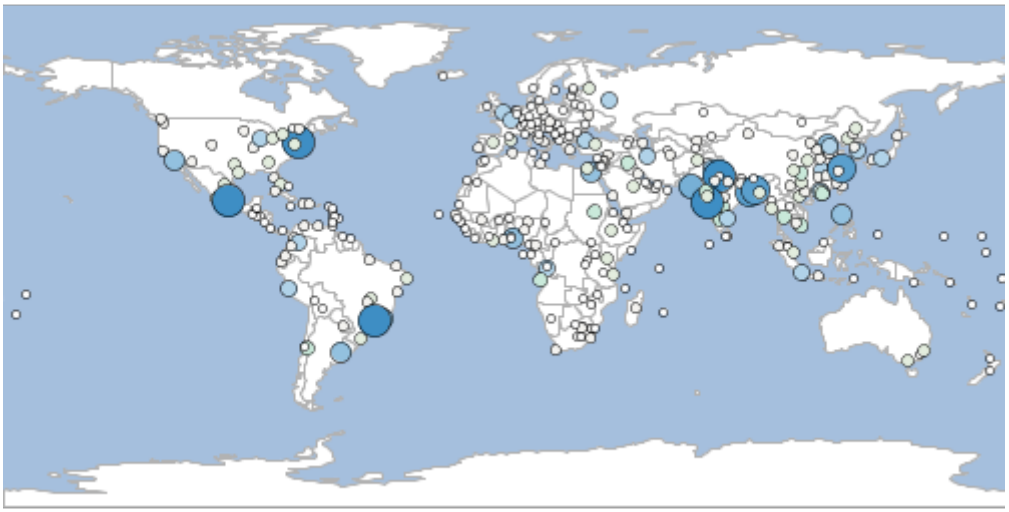
Create a Gradient Symbolizer from a Layer's Field using equal interval method

```
Workspace workspace = new GeoPackage('src/main/resources/data.gpkg')
Layer countries = workspace.get("countries")
Gradient gradient = new Gradient(countries, "PEOPLE", "equalinterval", 3, "Reds")
countries.style = gradient
```



Create a custom Gradient Symbolizer between Symbolizers and values

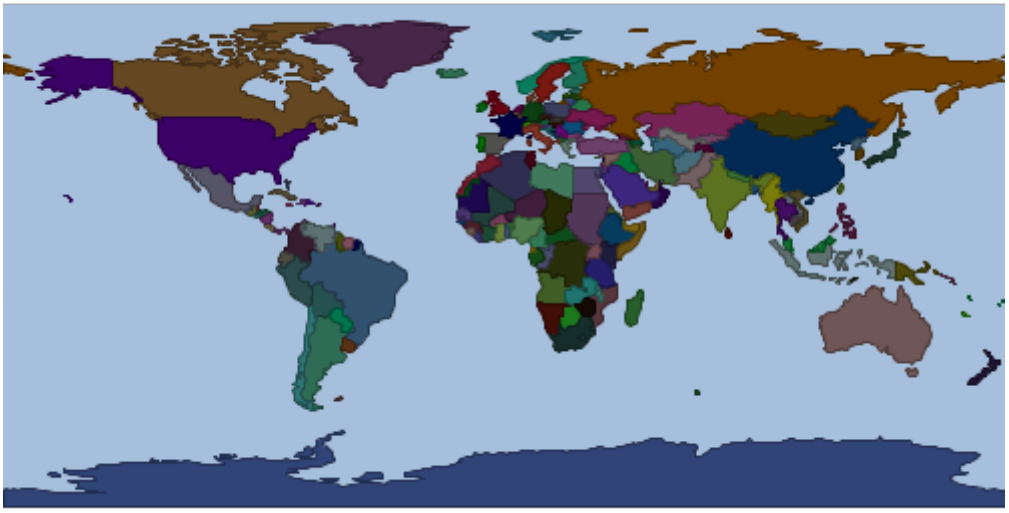
```
Gradient gradient = new Gradient(
        new Property("POP2020"),
        [0, 10000, 20000, 30000],
        [
                new Shape("white", 4).stroke("black", 0.5),
                new Shape("#b0d2e8", 8).stroke("black", 0.5),
                new Shape("#3e8ec4", 16).stroke("black", 0.5),
                new Shape("#08306b", 24).stroke("black", 0.5)
        ],
        5,
        "linear"
)
```



Creating Unique Values

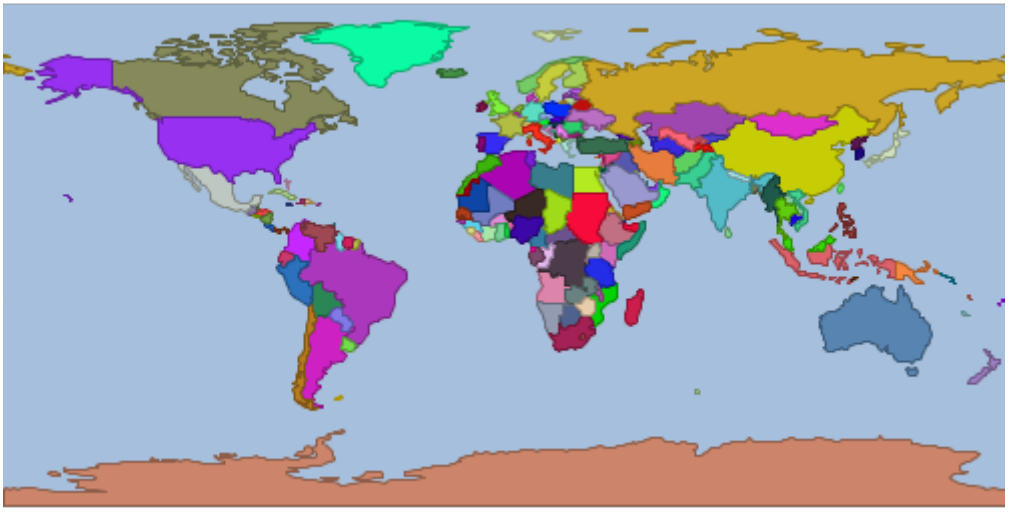
Create a Unique Values Symbolizer from a Layer's Field

```
Workspace workspace = new GeoPackage('src/main/resources/data.gpkg')
Layer countries = workspace.get("countries")
UniqueValues uniqueValues = new UniqueValues(countries, "NAME")
countries.style = uniqueValues
```



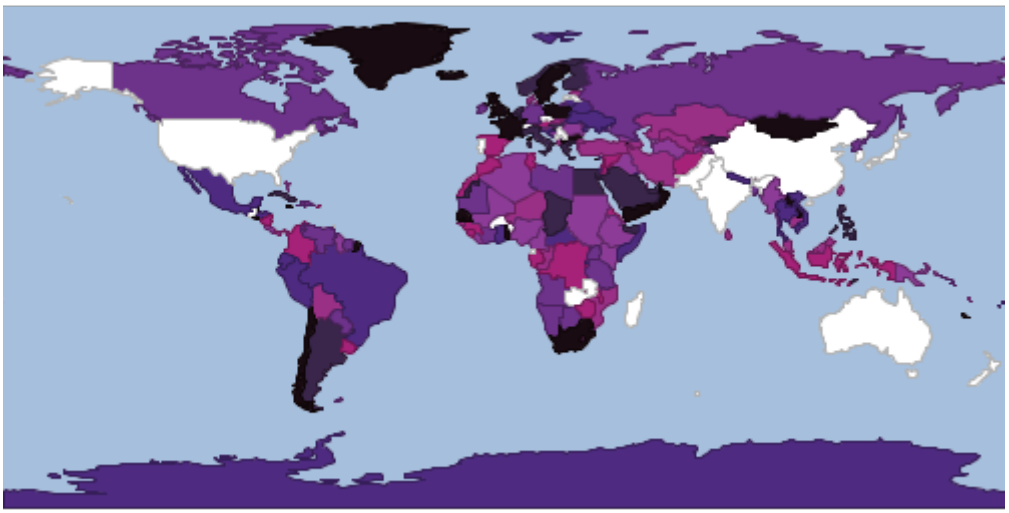
Create a Unique Values Symbolizer from a Layer's Field and a Closure

```
Workspace workspace = new GeoPackage('src/main/resources/data.gpkg')
Layer countries = workspace.get("countries")
UniqueValues uniqueValues = new UniqueValues(countries, "NAME", {int index, String
value -> Color.getRandom()})
countries.style = uniqueValues
```



Create a Unique Values Symbolizer from a Layer's Field and a color palette

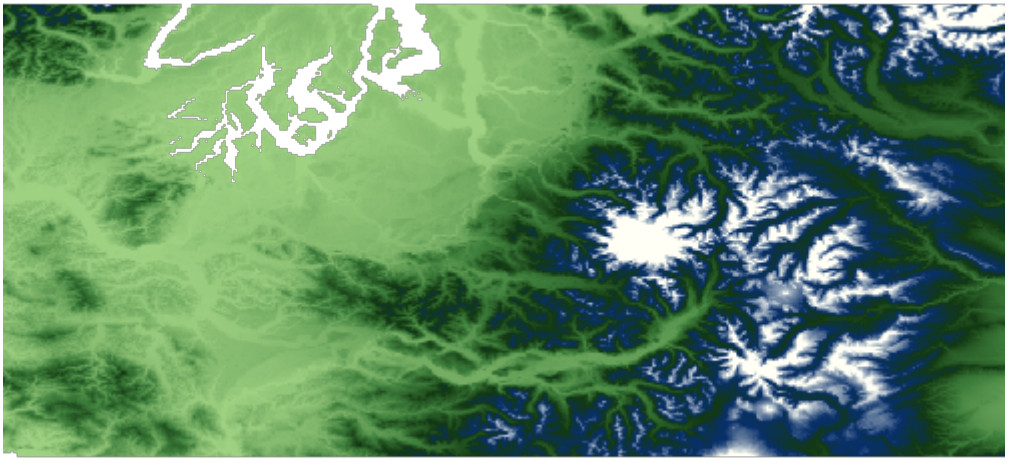
```
Workspace workspace = new GeoPackage('src/main/resources/data.gpkg')
Layer countries = workspace.get("countries")
UniqueValues uniqueValues = new UniqueValues(countries, "NAME",
"LightPurpleToDarkPurpleHeatMap")
countries.style = uniqueValues
```



Creating Color Maps

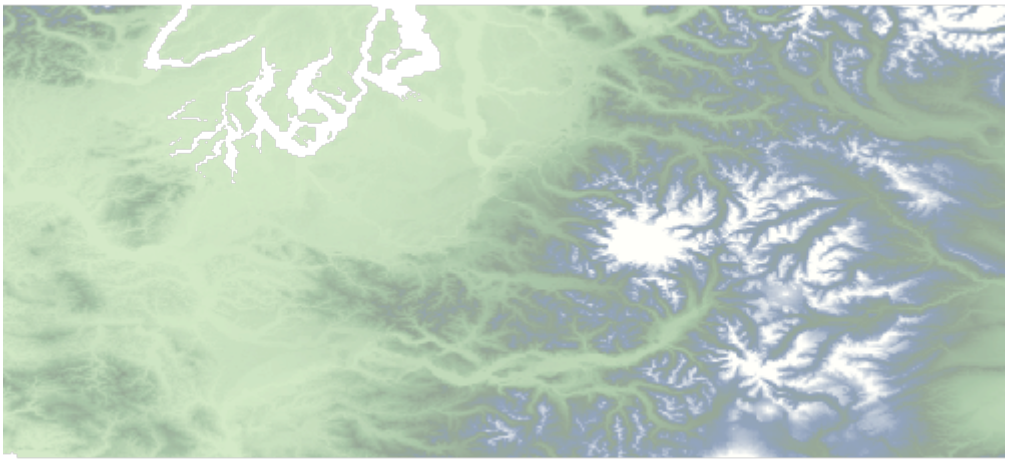
Create a ColorMap Symbolizer for a Raster using a list of Colors

```
Format format = new GeoTIFF(new File('src/main/resources/pc.tif'))
Raster raster = format.read()
ColorMap colorMap = new ColorMap([
        [color: "#9fd182", quantity:25],
        [color: "#3e7f3c", quantity:470],
        [color: "#133912", quantity:920],
        [color: "#08306b", quantity:1370],
        [color: "#fffff5", quantity:1820],
])
raster.style = colorMap
```



Create a ColorMap Symbolizer for a Raster using a list of Colors with opacity

```
Format format = new GeoTIFF(new File('src/main/resources/pc.tif'))
Raster raster = format.read()
ColorMap colorMap = new ColorMap([
        [color: "#9fd182", quantity:25],
        [color: "#3e7f3c", quantity:470],
        [color: "#133912", quantity:920],
        [color: "#08306b", quantity:1370],
        [color: "#fffff5", quantity:1820],
]).opacity(0.25)
raster.style = colorMap
```



Create a ColorMap Symbolizer for a Raster using a color palette

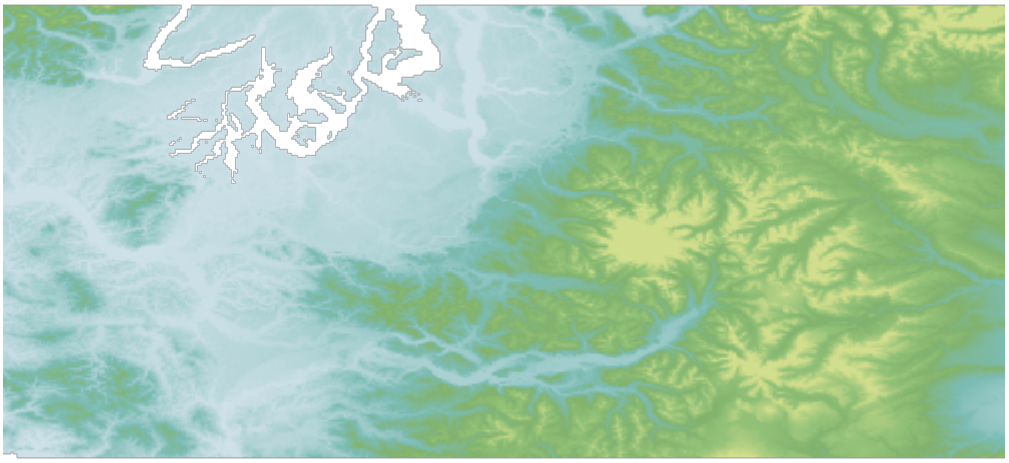
```
Format format = new GeoTIFF(new File('src/main/resources/pc.tif'))
Raster raster = format.read()
ColorMap colorMap = new ColorMap(
        25,             ①
        1820,           ②
        "MutedTerrain", ③
        5               ④
)
println colorMap
raster.style = colorMap
```

① min value

② max value

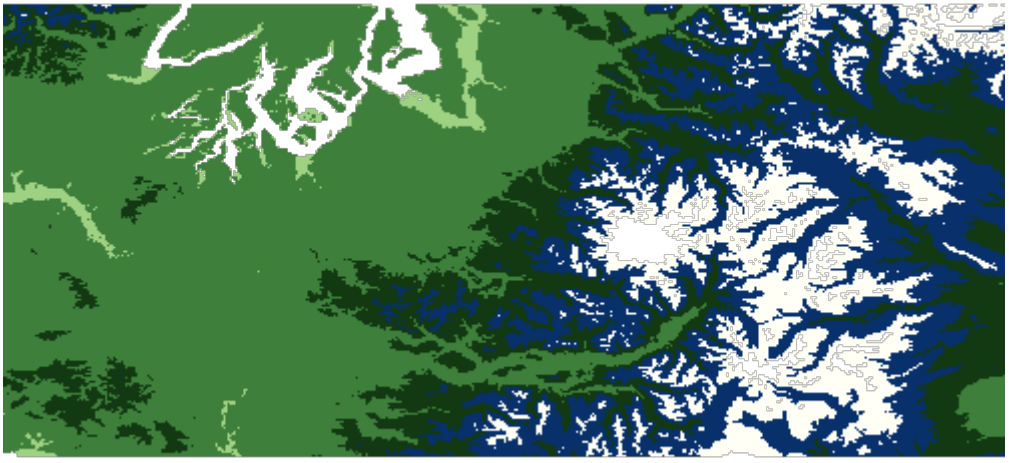
③ color palette name

④ number of categories



Create a ColorMap Symbolizer with intervals for a Raster using a list of Colors

```
Format format = new GeoTIFF(new File('src/main/resources/pc.tif'))
Raster raster = format.read()
ColorMap colorMap = new ColorMap([
        [color: "#9fd182", quantity:25],
        [color: "#3e7f3c", quantity:470],
        [color: "#133912", quantity:920],
        [color: "#08306b", quantity:1370],
        [color: "#fffff5", quantity:1820],
], "intervals", true)
raster.style = colorMap
```



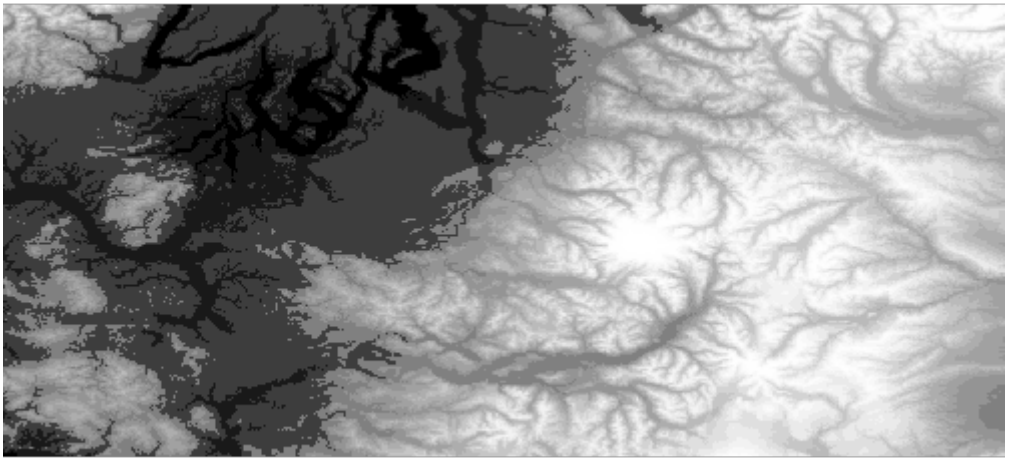
Creating Channel Selection and Contrast Enhancement

Create a Raster Symbolizer using ChannelSelection and ContrastEnhancement using the normalize method

```
Format format = new GeoTIFF(new File('src/main/resources/pc.tif'))
Raster raster = format.read()
Symbolizer symbolizer = new ChannelSelection()
        .gray("1", new ContrastEnhancement("normalize"))
raster.style = symbolizer
```

Create a Raster Symbolizer using ChannelSelection and ContrastEnhancement using the histogram method

```
Format format = new GeoTIFF(new File('src/main/resources/pc.tif'))
Raster raster = format.read()
Symbolizer symbolizer = new ChannelSelection()
        .gray("1", new ContrastEnhancement("histogram", 0.65))
raster.style = symbolizer
```



Reading and Writing Styles

Style Readers and Writers are found in the **geoscript.style.io** package.

Finding Style Readers and Writers

List all Style Writers

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

```
SLDWriter
ColorTableWriter
YSLDWriter
```

Find a Style Writer

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

```
SLDWriter
```

List all Style Readers

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

```
SLDReader
CSSReader
ColorTableReader
YSLDReader
SimpleStyleReader
```

Find a Style Reader

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

```
SLDReader
```

SLD

GeoScript Groovy can read and write Style Layer Descriptor (SLD) documents.

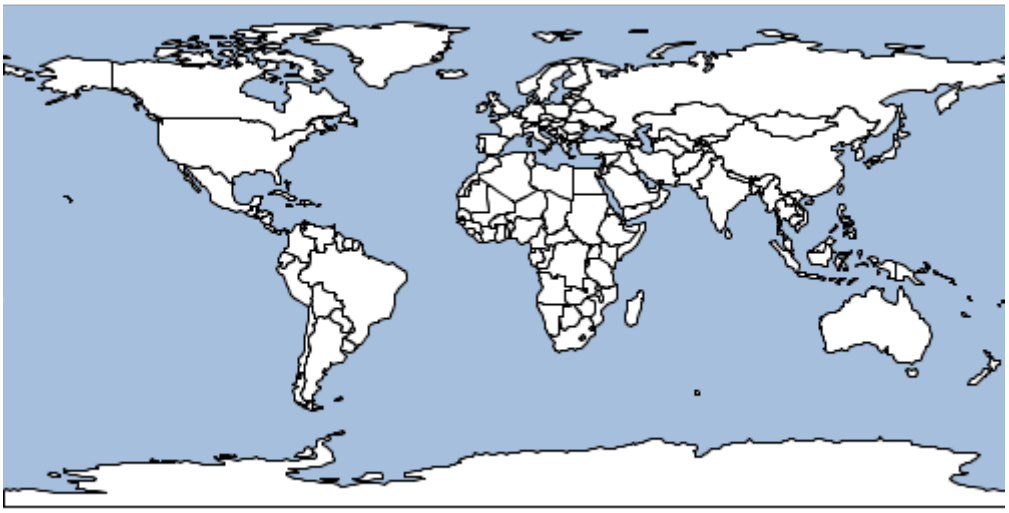
Write a Symbolizer to SLD

```
Symbolizer symbolizer = new Fill("white") + new Stroke("black", 0.5)
SLDWriter writer = new SLDWriter()
String sld = writer.write(symbolizer)
println sld
```

```
<?xml version="1.0" encoding="UTF-8"?>
<sld:StyledLayerDescriptor xmlns="http://www.opengis.net/sld"
xmlns:sld="http://www.opengis.net/sld" xmlns:gml="http://www.opengis.net/gml"
xmlns:ogc="http://www.opengis.net/ogc" version="1.0.0">
  <sld:UserLayer>
    <sld:LayerFeatureConstraints>
      <sld:FeatureTypeConstraint/>
    </sld:LayerFeatureConstraints>
    <sld:UserStyle>
      <sld:Name>Default Styler</sld:Name>
      <sld:FeatureTypeStyle>
        <sld:Name>name</sld:Name>
        <sld:Rule>
          <sld:PolygonSymbolizer>
            <sld:Fill>
              <sld:CssParameter name="fill">#ffffff</sld:CssParameter>
            </sld:Fill>
          </sld:PolygonSymbolizer>
          <sld:LineSymbolizer>
            <sld:Stroke>
              <sld:CssParameter name="stroke-width">0.5</sld:CssParameter>
            </sld:Stroke>
          </sld:LineSymbolizer>
        </sld:Rule>
      </sld:FeatureTypeStyle>
    </sld:UserStyle>
  </sld:UserLayer>
</sld:StyledLayerDescriptor>
```

Read a Style from an SLD String

```
        String sld = """<?xml version="1.0" encoding="UTF-8"?>
<sld:StyledLayerDescriptor xmlns="http://www.opengis.net/sld"
xmlns:sld="http://www.opengis.net/sld" xmlns:ogc="http://www.opengis.net/ogc"
xmlns:gml="http://www.opengis.net/gml" version="1.0.0">
  <sld:UserLayer>
    <sld:LayerFeatureConstraints>
      <sld:FeatureTypeConstraint/>
    </sld:LayerFeatureConstraints>
    <sld:UserStyle>
      <sld:Name>Default Styler</sld:Name>
      <sld:FeatureTypeStyle>
        <sld:Name>name</sld:Name>
        <sld:Rule>
          <sld:PolygonSymbolizer>
            <sld:Fill>
              <sld:CssParameter name="fill">#ffffff</sld:CssParameter>
            </sld:Fill>
          </sld:PolygonSymbolizer>
          <sld:LineSymbolizer>
            <sld:Stroke>
              <sld:CssParameter name="stroke">#000000</sld:CssParameter>
              <sld:CssParameter name="stroke-width">0.5</sld:CssParameter>
            </sld:Stroke>
          </sld:LineSymbolizer>
        </sld:Rule>
      </sld:FeatureTypeStyle>
    </sld:UserStyle>
  </sld:UserLayer>
</sld:StyledLayerDescriptor>
"""
        SLDReader reader = new SLDReader()
        Style style = reader.read(sld)

        Workspace workspace = new GeoPackage('src/main/resources/data.gpkg')
        Layer countries = workspace.get("countries")
        countries.style = style
```

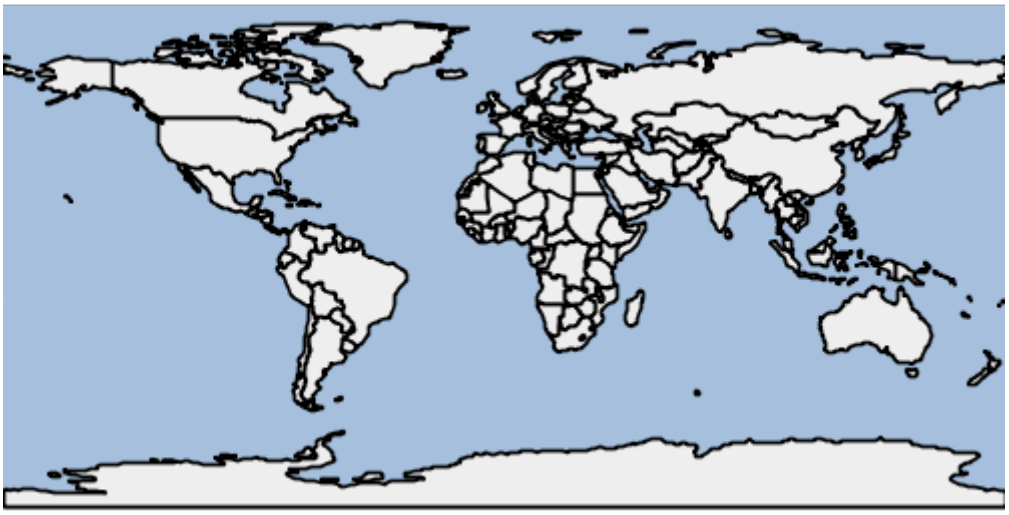


CSS

GeoScript Groovy can only read CSS documents.

Read a Style from an CSS String

```
        String css = """
* {
   fill: #eeeeee;
   fill-opacity: 1.0;
   stroke: #000000;
   stroke-width: 1.2;
}
"""
        CSSReader reader = new CSSReader()
        Style style = reader.read(css)

        Workspace workspace = new GeoPackage('src/main/resources/data.gpkg')
        Layer countries = workspace.get("countries")
        countries.style = style
```



YSLD

GeoScript Groovy can read and write YAML Style Layer Descriptors (YSLD) documents.

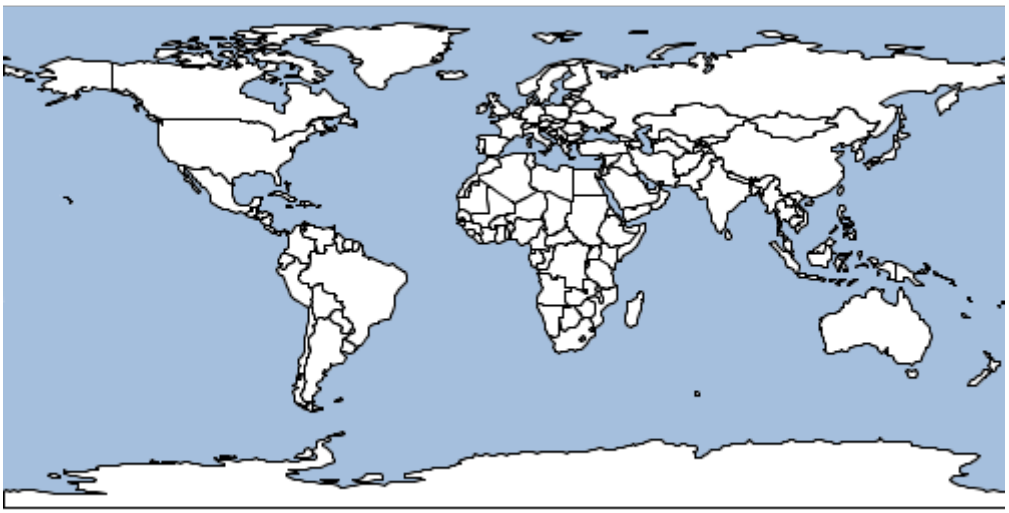
Write a Symbolizer to YSLD

```
Symbolizer symbolizer = new Fill("white") + new Stroke("black", 0.5)
YSLDWriter writer = new YSLDWriter()
String ysld = writer.write(symbolizer)
println ysld
```

```
name: Default Styler
feature-styles:
- name: name
  rules:
  - scale: [min, max]
    symbolizers:
    - polygon:
        fill-color: '#FFFFFF'
    - line:
        stroke-color: '#000000'
        stroke-width: 0.5
```

Read a Style from an YAML Style Layer Descriptors (YSLD) String

```
        String ysld = """
name: Default Styler
feature-styles:
- name: name
  rules:
  - scale: [min, max]
    symbolizers:
    - polygon:
        fill-color: '#FFFFFF'
    - line:
        stroke-color: '#000000'
        stroke-width: 0.5
"""
        YSLDReader reader = new YSLDReader()
        Style style = reader.read(ysld)

        Workspace workspace = new GeoPackage('src/main/resources/data.gpkg')
        Layer countries = workspace.get("countries")
        countries.style = style
```



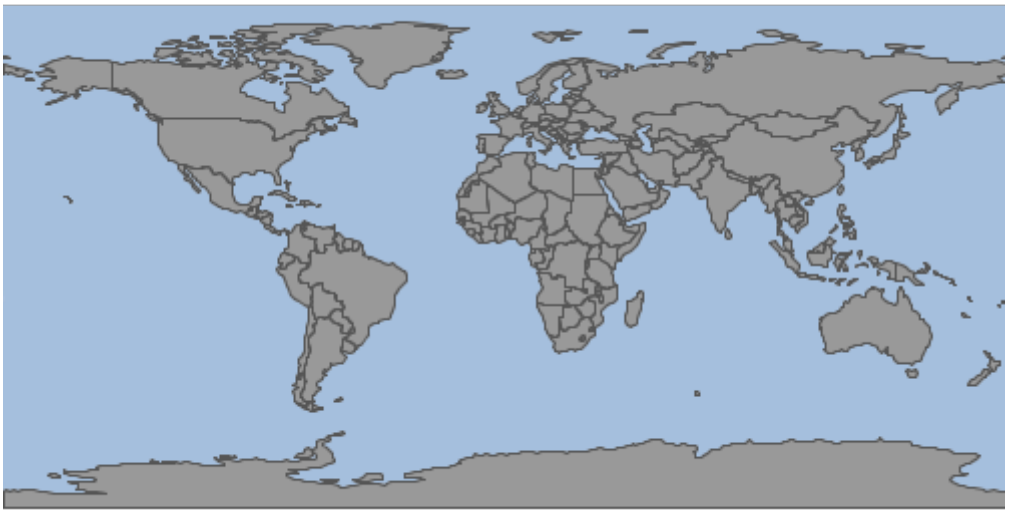
Simple Style Reader

A SimpleStyleReader that can easily create simple Styles using Maps or Strings.

- Fill properties: fill and fill-opacity
- Stroke properties: stroke, stroke-width, stroke-opacity
- Shape properties: shape, shape-size, shape-type
- Label properties: label-size, label-style, label-weight, label-family

Read a Style with fill and stroke properties from a Simple Style String

```
String str = "fill=#555555 fill-opacity=0.6 stroke=#555555 stroke-width=0.5"
SimpleStyleReader reader = new SimpleStyleReader()
Style style = reader.read(str)

Workspace workspace = new GeoPackage('src/main/resources/data.gpkg')
Layer countries = workspace.get("countries")
countries.style = style
```

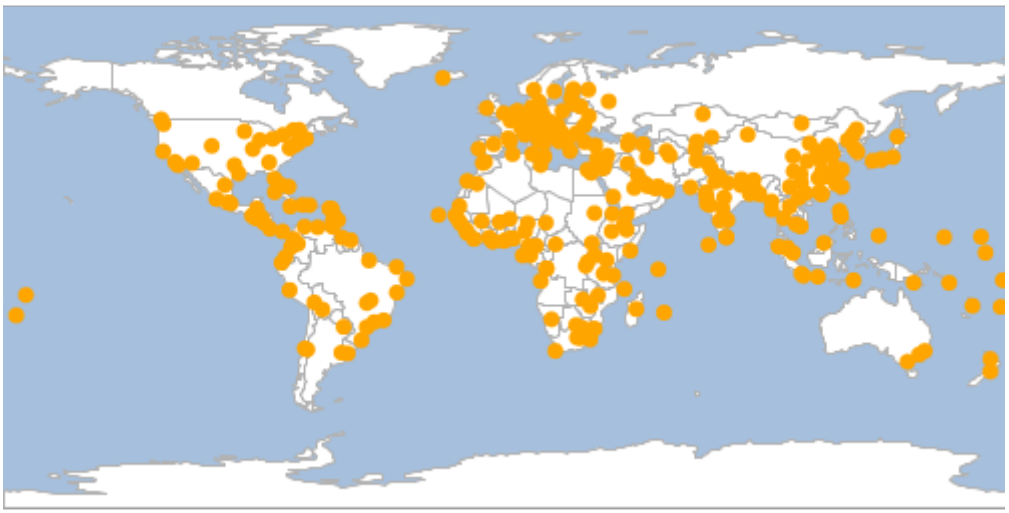


Read a Style with fill, stroke, and label properties from a Simple Style String

```
String str = "fill=white stroke=navy label=NAME label-size=10"
SimpleStyleReader reader = new SimpleStyleReader()
Style style = reader.read(str)

Workspace workspace = new GeoPackage('src/main/resources/data.gpkg')
Layer countries = workspace.get("countries")
countries.style = style
```

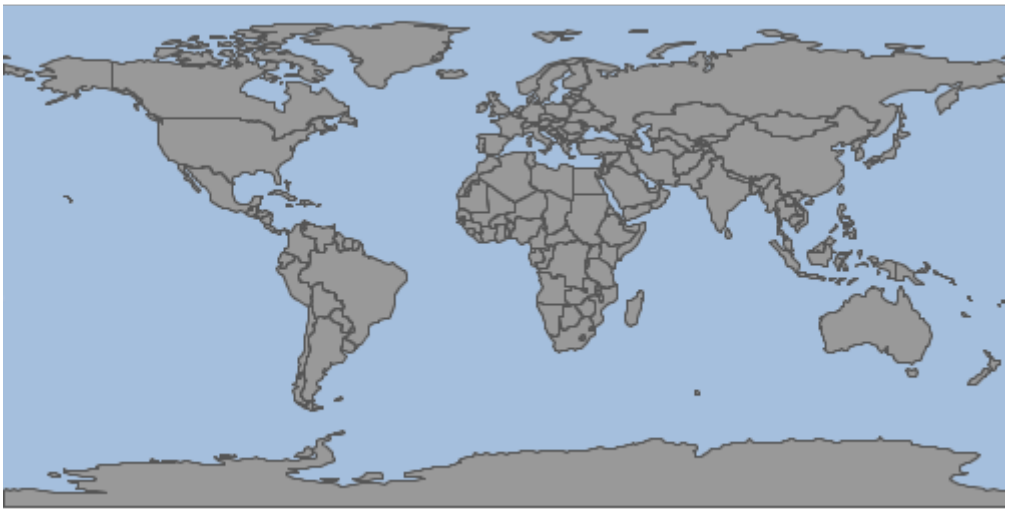
Read a Style with shape properties from a Simple Style String

```
String str = "shape-type=circle shape-size=8 shape=orange"
SimpleStyleReader reader = new SimpleStyleReader()
Style style = reader.read(str)
println style

Workspace workspace = new GeoPackage('src/main/resources/data.gpkg')
Layer places = workspace.get("places")
places.style = style
```



Read a Style with fill and stroke properties from a Simple Style Map

```
Map map = [
        'fill': '#555555',
        'fill-opacity': 0.6,
        'stroke': '#555555',
        'stroke-width': 0.5
]
SimpleStyleReader reader = new SimpleStyleReader()
Style style = reader.read(map)

Workspace workspace = new GeoPackage('src/main/resources/data.gpkg')
Layer countries = workspace.get("countries")
countries.style = style
```



Color Table

GeoScript Groovy can read and write color table strings and files. This format can be used with ColorMaps to style Rasters.

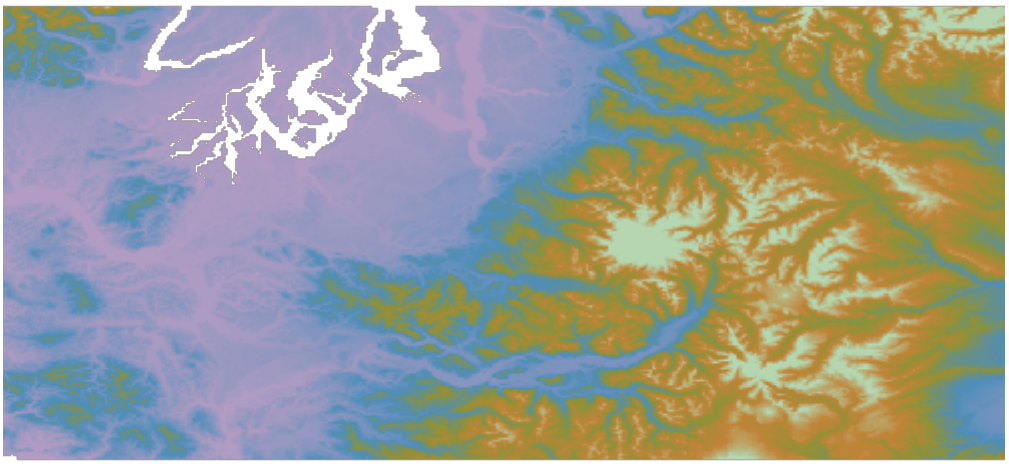
Write a ColorMap to a color table string

```
ColorMap colorMap = new ColorMap(25, 1820, "BoldLandUse", 5)
ColorTableWriter writer = new ColorTableWriter()
String str = writer.write(colorMap)
println str
```

```
25.0 178 156 195
473.75 79 142 187
922.5 143 146 56
1371.25 193 132 55
1820.0 181 214 177
```

Read a ColorMap from a color table string

```
        Format format = new GeoTIFF(new File('src/main/resources/pc.tif'))
        Raster raster = format.read()
        ColorTableReader reader = new ColorTableReader()
        ColorMap colorMap = reader.read("""25.0 178 156 195
473.75 79 142 187
922.5 143 146 56
1371.25 193 132 55
1820.0 181 214 177
""")
        raster.style = colorMap
```



Workspace Recipes

The Workspace classes are in the **geoscript.workspace** package.

A Workspace is a collection of Layers. You can create, add, remove, and get Layers. There are many different kinds of Workspaces in GeoScript including Memory, PostGIS, Directory (for Shapefiles), GeoPackage, and many more.

Using Workspaces

Create a Workspace

```
Workspace workspace = new Workspace()
```

Create a Layer

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

```
cities
```

Check whether a Workspace has a Layer by name

```
boolean hasCities = workspace.has("cities")
println hasCities
```

```
true
```

Get a Layer from a Workspace

```
Layer citiesLayer = workspace.get('cities')
println citiesLayer
```

```
cities
```

Add a Layer to a Workspace

```
Schema statesSchema = new Schema("states", [
        new Field("geom", "Polygon", "EPSG:4326"),
        new Field("id", "Integer"),
        new Field("name", "String")
])
Layer statesLayer = new Layer("states", statesSchema)
workspace.add(statesLayer)
println workspace.has("states")
```

```
true
```

Get the names of all Layers in a Workspace

```
List<String> names = workspace.names
names.each { String name ->
    println name
}
```

```
PostGIS (JNDI)
SpatiaLite (JNDI)
Shapefile
MySQL (JNDI)
PostGIS
H2 (JNDI)
H2
Properties
MySQL
Geobuf
Web Feature Server (NG)
GeoPackage
Directory of spatial files (shapefiles)
```

Remove a Layer from a Workspace

```
workspace.remove("cities")
println workspace.has('cities')
```

```
false
```

Close the Workspace when you are done

```
workspace.close()
```

Using a Directory Workspace

A Directory Workspace is a directory of Shapefiles.

Create a Directory Workspace

```
Directory directory = new Directory("src/main/resources/data")
println directory.toString()
```

```
Directory[/home/travis/build/jericks/geoscript-groovy-
cookbook/src/main/resources/data]
```

View the Workspace's format

```
String format = directory.format
println format
```

```
Directory
```

View the Workspace's File

```
File file = directory.file
println file
```

```
/home/travis/build/jericks/geoscript-groovy-cookbook/src/main/resources/data
```

View the Workspace's list of Layer names

```
List names = directory.names
names.each { String name ->
    println name
}
```

```
states
```

Get a Layer by name

```
Layer layer = directory.get("states")
int count = layer.count
println "Layer ${layer.name} has ${count} Features."
```

```
Layer states has 49 Features.
```

Close the Directory when done.

```
directory.close()
```

Investigating Workspaces

Get available Workspace names

```
List<String> names = Workspace.getWorkspaceNames()
names.each { String name ->
    println name
}
```

```
PostGIS (JNDI)
SpatiaLite (JNDI)
Shapefile
MySQL (JNDI)
PostGIS
H2 (JNDI)
H2
Properties
MySQL
Geobuf
Web Feature Server (NG)
GeoPackage
Directory of spatial files (shapefiles)
```

Get parameters for a Workspace

```
List<Map> parameters = Workspace.getWorkspaceParameters("GeoPackage")
parameters.each { Map param ->
    println "Parameter = ${param.key} Type = ${param.type} Required?
${param.required}"
}
```

```
Parameter = dbtype Type = java.lang.String Required? true
Parameter = database Type = java.io.File Required? true
Parameter = user Type = java.lang.String Required? false
Parameter = passwd Type = java.lang.String Required? false
Parameter = namespace Type = java.lang.String Required? false
Parameter = Expose primary keys Type = java.lang.Boolean Required? false
Parameter = max connections Type = java.lang.Integer Required? false
Parameter = min connections Type = java.lang.Integer Required? false
Parameter = fetch size Type = java.lang.Integer Required? false
Parameter = Batch insert size Type = java.lang.Integer Required? false
Parameter = Connection timeout Type = java.lang.Integer Required? false
Parameter = validate connections Type = java.lang.Boolean Required? false
Parameter = Test while idle Type = java.lang.Boolean Required? false
Parameter = Evictor run periodicity Type = java.lang.Integer Required? false
Parameter = Max connection idle time Type = java.lang.Integer Required? false
Parameter = Evictor tests per run Type = java.lang.Integer Required? false
Parameter = Primary key metadata table Type = java.lang.String Required? false
Parameter = Session startup SQL Type = java.lang.String Required? false
Parameter = Session close-up SQL Type = java.lang.String Required? false
Parameter = Callback factory Type = java.lang.String Required? false
```

Creating Workspaces

Creating a Workspace from a connection string

You can create a Workspace from a connection string that contains paramters in key=value format with optional single quotes.

Create a Shapefile Workspace

```
String connectionString = "url='states.shp' 'create spatial index'=true"
Workspace workspace = Workspace.getWorkspace(connectionString)
```

Create a GeoPackage Workspace

```
connectionString = "dbtype=geopkg database=layers.gpkg"
workspace = Workspace.getWorkspace(connectionString)
```

Create a H2 Workspace

```
connectionString = "dbtype=h2 database=layers.db"
workspace = Workspace.getWorkspace(connectionString)
```

Creating a Workspace from a connection map

You can create a Workspace from a connection map that contains paramters.

Create a H2 Workspace

```
Map params = [dbtype: 'h2', database: 'test.db']
Workspace workspace = Workspace.getWorkspace(params)
```

Create a PostGIS Workspace

```
params = [
    dbtype: 'postgis',
    database: 'postgres',
    host: 'localhost',
    port: 5432,
    user: 'postgres',
    passwd: 'postgres'
]
workspace = Workspace.getWorkspace(params)
```

Create a GeoBuf Workspace

```
params = [file: 'layers.pbf', precision: 6, dimension:2]
workspace = Workspace.getWorkspace(params)
```

Creating Directory Workspaces

Create a Directory Workspace from a directory name

```
Workspace workspace = new Directory("src/main/resources/shapefiles")
println workspace.format
println "---------"
workspace.names.each { String name ->
    println "${name} (${workspace.get(name).count})"
}
```

```
Directory
---------
ocean (2)
countries (177)
```

Create a Directory Workspace from a File directory

```
Workspace workspace = new Directory(new File("src/main/resources/shapefiles"))
println workspace.format
println "---------"
workspace.names.each { String name ->
    println "${name} (${workspace.get(name).count})"
}
```

```
Directory
---------
ocean (2)
countries (177)
```

Create a Directory Workspace from a URL

```
Directory directory = Directory.fromURL(
    new URL
("http://www.naturalearthdata.com/http//www.naturalearthdata.com/download/110m/cultura
l/ne_110m_admin_0_countries.zip"),
    new File("naturalearth")
)
println directory.format
println "---------"
directory.names.each { String name ->
    println "${name} (${directory.get(name).count})"
}
```

```
Directory
---------
ne_110m_admin_0_countries (177)
```

Creating GeoPackage Workspaces

Create a GeoPackage Workspace from a file name

```
Workspace workspace = new GeoPackage("src/main/resources/data.gpkg")
println workspace.format
println "----------"
workspace.names.each { String name ->
    println "${name} (${workspace.get(name).count})"
}
```

```
GeoPackage
----------
countries (177)
ocean (2)
places (326)
rivers (460)
states (52)
```

Create a GeoPackage Workspace from a File

```
Workspace workspace = new GeoPackage(new File("src/main/resources/data.gpkg"))
println workspace.format
println "----------"
workspace.names.each { String name ->
    println "${name} (${workspace.get(name).count})"
}
```

```
GeoPackage
----------
countries (177)
ocean (2)
places (326)
rivers (460)
states (52)
```

Creating H2 Workspaces

Create a H2 Workspace from a File

```
Workspace workspace = new H2(new File("src/main/resources/h2/data.db"))
println workspace.format
println "--"
workspace.names.each { String name ->
    println "${name} (${workspace.get(name).count})"
}
```

```
H2
--
countries (177)
ocean (2)
places (326)
states (52)
```

Creating Geobuf Workspaces

Create a Geobuf Workspace from a File

```
Workspace workspace = new Geobuf(new File("src/main/resources/geobuf"))
println workspace.format
println "------"
workspace.names.each { String name ->
    println "${name} (${workspace.get(name).count})"
}
```

```
Geobuf
------
countries (177)
ocean (2)
places (326)
```

Creating Property Workspaces

Create a Property Workspace from a File

```
Workspace workspace = new Property(new File("src/main/resources/property"))
println workspace.format
println "--------"
workspace.names.each { String name ->
    println "${name} (${workspace.get(name).count})"
}
```

```
Property
------
circles (10)
places (10)
```

Layer Recipes

The Layer classes are in the **geoscript.layer** package.

A Layer is a collection of Features.

Getting a Layer's Properties

Get a Layer from a Workspace and it's name

```
Workspace workspace = new GeoPackage("src/main/resources/data.gpkg")
Layer layer = workspace.get("countries")
String name = layer.name
println "Name: ${name}"
```

```
Name: countries
```

The Layer's Format

```
String format = layer.format
println "Format: ${format}"
```

```
Format: GeoPackage
```

Count the number of Features

```
int count = layer.count
println "# of Features: ${count}"
```

```
# of Features: 177
```

Get the Layer's Projection

```
Projection proj = layer.proj
println "Projection: ${proj}"
```

```
Projection: EPSG:4326
```

Get the Bounds of the Layer

```
Bounds bounds = layer.bounds
println "Bounds: ${bounds}"
```

```
Bounds: (-179.99999999999997,-
90.00000000000003,180.00000000000014,83.64513000000002,EPSG:4326)
```

Getting a Layer's Features

Iterate over a Layer's Features

```
Workspace workspace = new GeoPackage("src/main/resources/data.gpkg")
Layer layer = workspace.get("states")
layer.eachFeature { Feature feature ->
    println feature["NAME_1"]
}
```

```
Minnesota
Montana
North Dakota
Hawaii
Idaho
Washington
Arizona
California
Colorado
Nevada
...
```

Iterate over a subset of a Layer's Features

```
Workspace workspace = new GeoPackage("src/main/resources/data.gpkg")
Layer layer = workspace.get("states")
layer.eachFeature("NAME_1 LIKE 'M%'") { Feature feature ->
    println feature["NAME_1"]
}
```

```
Minnesota
Montana
Missouri
Massachusetts
Mississippi
Maryland
Maine
Michigan
```

Iterate over a Layer's Features with parameters.

```
Workspace workspace = new GeoPackage("src/main/resources/data.gpkg")
Layer layer = workspace.get("states")
layer.eachFeature(sort: ["NAME_1"], start: 0, max: 5, fields: ["NAME_1"], filter:
"NAME_1 LIKE 'M%'") { Feature feature ->
    println feature["NAME_1"]
}
```

```
Maine
Maryland
Massachusetts
Michigan
Minnesota
```

Parameters

- filter: The Filter or Filter String to limit the Features. Defaults to null.
- sort: A List of Lists that define the sort order [[Field or Field name, "ASC" or "DESC"],...]. Not all Layers support sorting!
- max: The maximum number of Features to include
- start: The index of the record to start the cursor at. Together with maxFeatures this simulates paging. Not all Layers support the start index and paging!
- fields: A List of Fields or Field names to include. Used to select only a subset of Fields.

Read all Feature into a List

```
Workspace workspace = new GeoPackage("src/main/resources/data.gpkg")
Layer layer = workspace.get("states")
List<Feature> features = layer.features

println "# Features = ${features.size()}"
features.each { Feature feature ->
    println feature["NAME_1"]
}
```

```
# Features = 52
Minnesota
Montana
North Dakota
Hawaii
Idaho
Washington
Arizona
California
Colorado
Nevada
...
```

Collect values from a Layer's Features

```
Workspace workspace = new GeoPackage("src/main/resources/data.gpkg")
Layer layer = workspace.get("states")
List<String> names = layer.collectFromFeature { Feature f ->
    f["NAME_1"]
}.sort()

println "# Names = ${names.size()}"
names.each { String name ->
    println name
}
```

```
# Names = 52
Alabama
Alaska
Arizona
Arkansas
California
Colorado
Connecticut
Delaware
District of Columbia
Florida
...
```

Collect values from a Layer's Features with parameters.

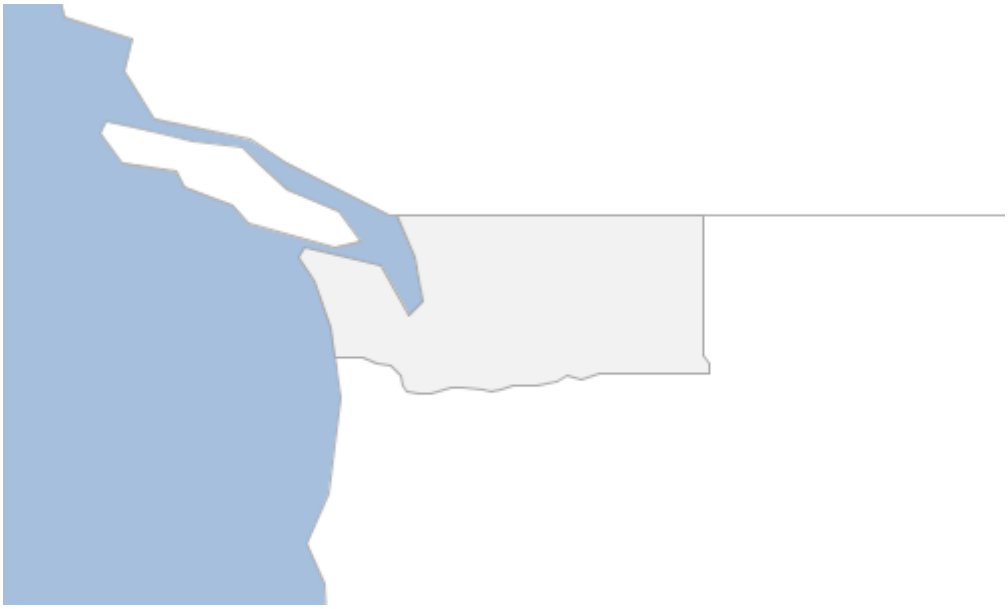
```
Workspace workspace = new GeoPackage("src/main/resources/data.gpkg")
Layer layer = workspace.get("states")
List<String> names = layer.collectFromFeature(
    sort: ["NAME_1"],
    start: 0,
    max: 5,
    fields: ["NAME_1"],
    filter: "NAME_1 LIKE 'M%'") { Feature f ->
        f["NAME_1"]
    }

println "# Names = ${names.size()}"
names.each { String name ->
    println name
}
```

```
# Names = 5
Maine
Maryland
Massachusetts
Michigan
Minnesota
```

Get the first Feature that matches the Filter.

```
Workspace workspace = new GeoPackage("src/main/resources/data.gpkg")
Layer layer = workspace.get("states")
Feature feature = layer.first(filter: "NAME_1='Washington'")
println feature.get("NAME_1")
```

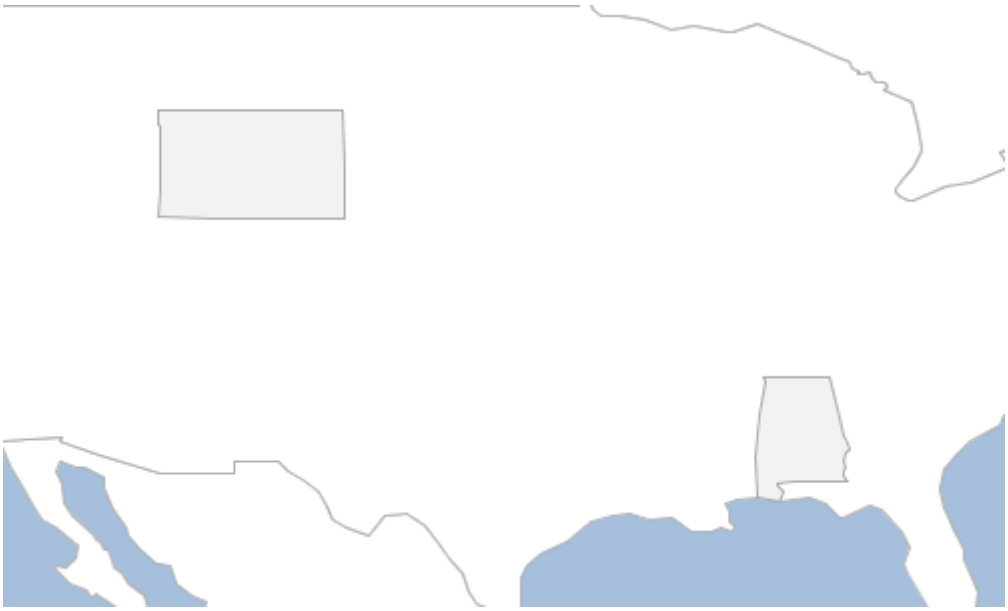
```
Washington
```



Get the first Feature sorted by name ascending and descending.

```
Workspace workspace = new GeoPackage("src/main/resources/data.gpkg")
Layer layer = workspace.get("states")

Feature featureAsc = layer.first(sort: "NAME_1 ASC")
println featureAsc.get("NAME_1")

Feature featureDesc = layer.first(sort: "NAME_1 DESC")
println featureDesc.get("NAME_1")
```

```
Alabama
Wyoming
```



Create a new Layer from an existing Layer with just the Features that match a Filter.

```
Workspace workspace = new Directory("target")
Layer layer = workspace.get("countries")
Layer disputedLayer = layer.filter("TYPE='Disputed'")
```

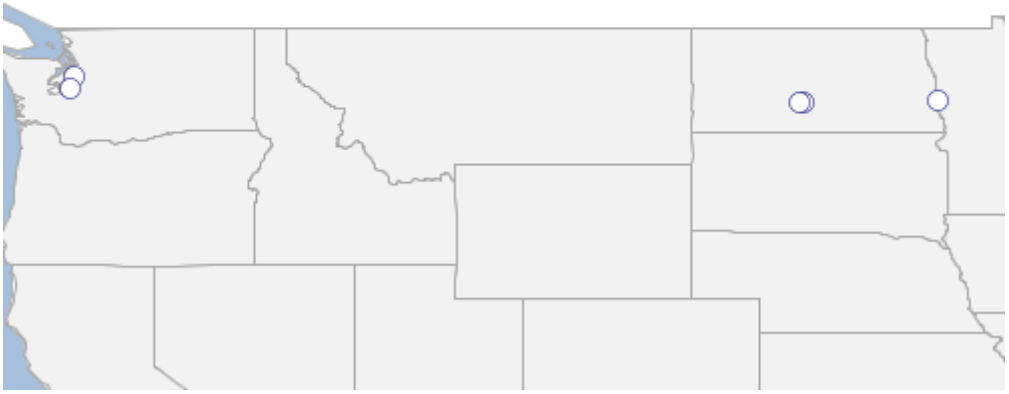
Adding, Updating, and Deleting

Add Features to a Layer

```
Workspace workspace = new Memory()
Schema schema = new Schema("cities", [
        new Field("geom", "Point", "EPSG:4326"),
        new Field("id", "Integer"),
        new Field("name", "String"),
        new Field("state", "String")
])
Layer layer = workspace.create(schema)

// Add a Feature with a Map
Map attributes = [
        geom: new Point(-122.333056, 47.609722),
        id: 1,
        name: "Seattle",
        state: "WA"
]
layer.add(attributes)

// Add a Feature with a List
List values = [
        new Point(-122.459444, 47.241389),
        2,
        "Tacoma",
```

```
        "WA"
]
layer.add(values)

// Add a Feature
Feature feature = schema.feature([
        id:3,
        name: "Fargo",
        state: "ND",
        geom: new Point(-96.789444, 46.877222)
])
layer.add(feature)

// Add Features from a List of Maps
List<Map> features = [
        [
                geom: new Point(-100.778889, 46.813333),
                id:4,
                name: "Bismarck",
                state: "ND"
        ],
        [
                geom: new Point(-100.891111, 46.828889),
                id: 5,
                name: "Mandan",
                state: "ND"
        ]
]
layer.add(features)
```

id	name	state
1	Seattle	WA
2	Tacoma	WA
3	Fargo	ND
4	Bismarck	ND
5	Mandan	ND

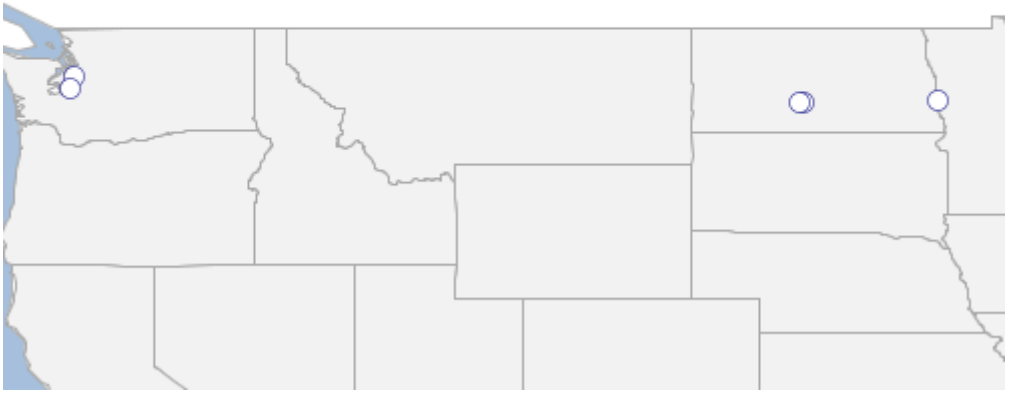


Update Features in a Layer

```
Workspace workspace = new Memory()
Schema schema = new Schema("cities", [
        new Field("geom", "Point", "EPSG:4326"),
        new Field("id", "Integer"),
        new Field("name", "String"),
        new Field("state", "String")
])
Layer layer = workspace.create(schema)
List<Map> features = [
        [
                geom: new Point(-122.333056, 47.609722),
                id: 1,
                name: "Seattle",
                state: "WA"
        ],
        [
                geom: new Point(-122.459444, 47.241389),
                id: 2,
                name: "Tacoma",
                state: "WA"
        ],
        [
                id:3,
                name: "Fargo",
                state: "ND",
                geom: new Point(-96.789444, 46.877222)
        ],
        [
                geom: new Point(-100.778889, 46.813333),
                id:4,
                name: "Bismarck",
                state: "ND"
        ],
        [
                geom: new Point(-100.891111, 46.828889),
                id: 5,
                name: "Mandan",
                state: "ND"
        ]
]
layer.add(features)

layer.update(layer.schema.state, "North Dakota", "state='ND'")
layer.update(layer.schema.state, "Washington", "state='WA'")
```

id	name	state
1	Seattle	Washington

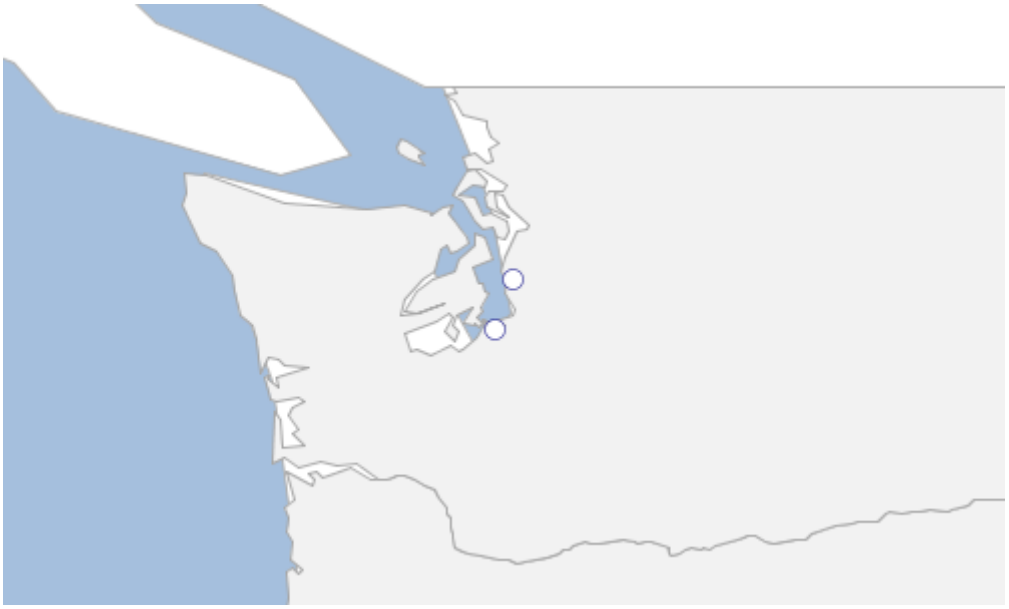
id	name	state
2	Tacoma	Washington
3	Fargo	North Dakota
4	Bismarck	North Dakota
5	Mandan	North Dakota



Delete Features from a Layer

```
Workspace workspace = new Memory()
Schema schema = new Schema("cities", [
        new Field("geom", "Point", "EPSG:4326"),
        new Field("id", "Integer"),
        new Field("name", "String"),
        new Field("state", "String")
])
Layer layer = workspace.create(schema)
List<Map> features = [
        [
                geom: new Point(-122.333056, 47.609722),
                id: 1,
                name: "Seattle",
                state: "WA"
        ],
        [
                geom: new Point(-122.459444, 47.241389),
                id: 2,
                name: "Tacoma",
                state: "WA"
        ],
        [
                id:3,
                name: "Fargo",
                state: "ND",
                geom: new Point(-96.789444, 46.877222)
        ],
        [
                geom: new Point(-100.778889, 46.813333),
                id:4,
                name: "Bismarck",
                state: "ND"
        ],
        [
                geom: new Point(-100.891111, 46.828889),
                id: 5,
                name: "Mandan",
                state: "ND"
        ]
]
layer.add(features)

layer.delete("state='ND'")
```

id	name	state
1	Seattle	WA
2	Tacoma	WA



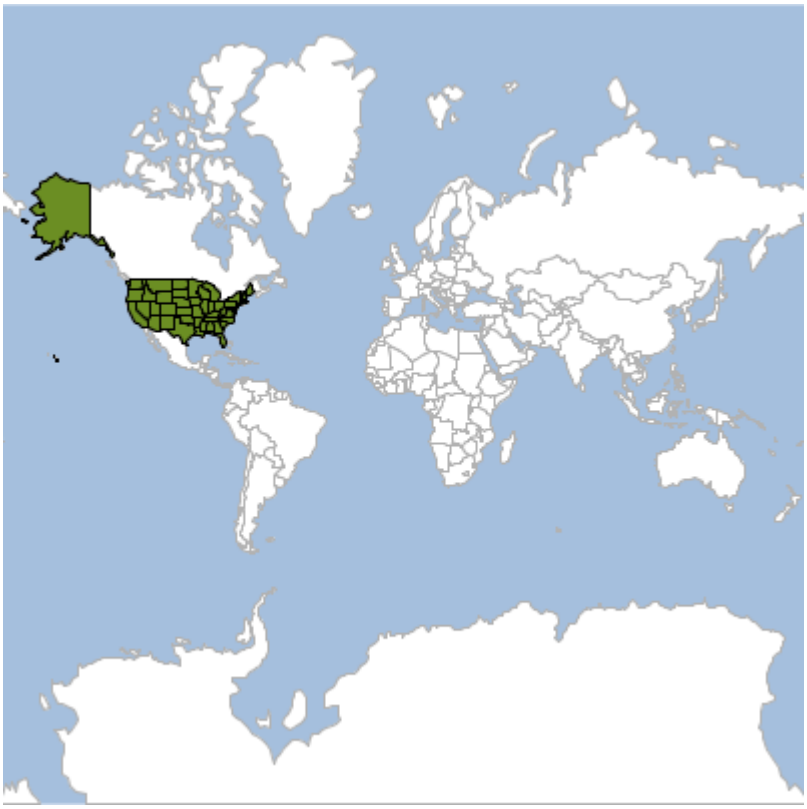
Geoprocessing

Reproject

Reproject a Layer from it's source projection to a target projection

```
Workspace workspace = new GeoPackage('src/main/resources/data.gpkg')
Layer states = workspace.get("states")
println "States = ${states.proj}"

Projection projection = new Projection("EPSG:3857")
Workspace outputWorkspace = new Memory()
Layer statesInWebMercator = states.reproject(projection, outputWorkspace,
"states_3857")
println "Reprojected States = ${statesInWebMercator.proj}"
```

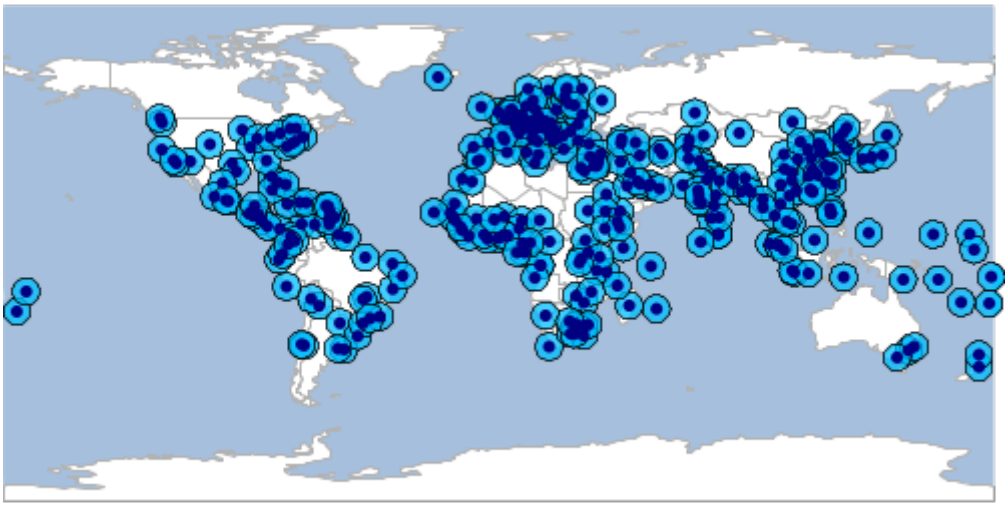
```
States = EPSG:4326
Reprojected States = EPSG:3857
```



Buffer

Buffer a Layer of populated places

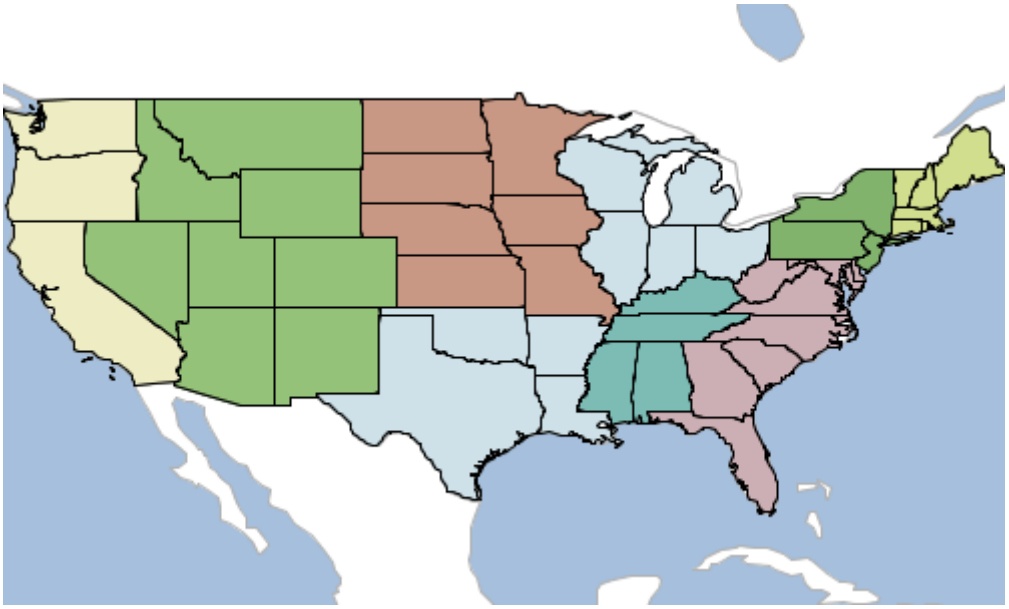
```
Workspace workspace = new GeoPackage(new File("src/main/resources/data.gpkg"))
Layer places = workspace.get("places")
Layer buffer = places.buffer(5)
```



Dissolve

Dissolve a Layer by a Field

```
Workspace workspace = new Directory(new File("src/main/resources/data"))
Layer states = workspace.get("states")
Layer regions = states.dissolve(states.schema.get("SUB_REGION"))
```

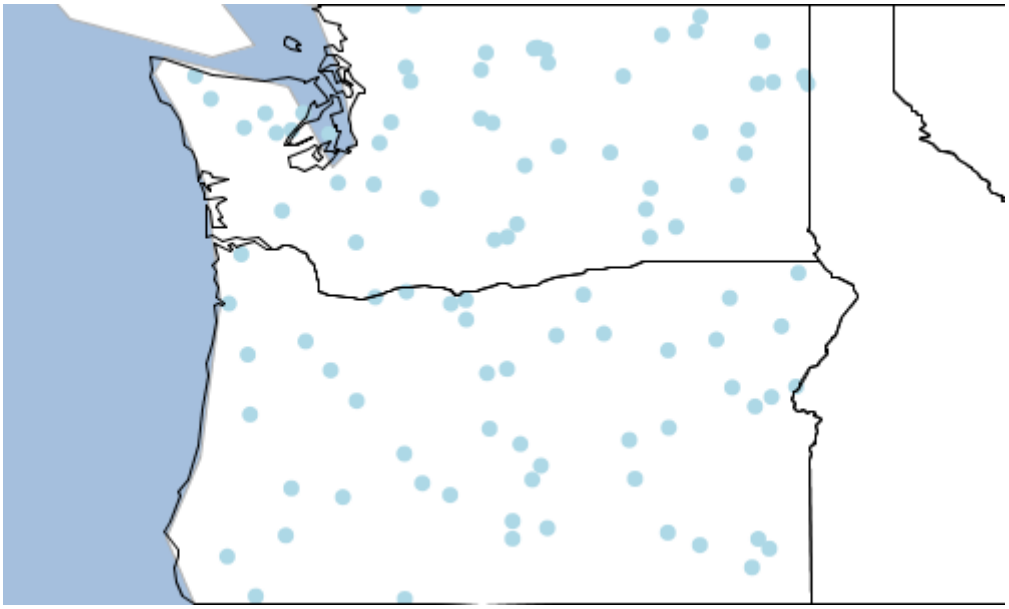


Merge

Merge two Layer together

```
Workspace workspace = new Directory(new File("src/main/resources/data"))
Layer states = workspace.get("states")
Feature washington = states.first(filter: "STATE_NAME='Washington'")
Feature oregon = states.first(filter: "STATE_NAME='Oregon'")

Schema waSchema = new Schema("washington",[
        new Field("geom","Point","EPSG:4326"),
        new Field("id","int")
])
Layer waLayer = new Memory().create(waSchema)
waLayer.withWriter { geoscript.layer.Writer writer ->
    Geometry.createRandomPoints(washington.geom, 50).points.eachWithIndex { Point pt,
int index ->
        writer.add(waSchema.feature([geom: pt, id: index]))
    }
}
println "The Washington Layer  has ${waLayer.count} features"

Schema orSchema = new Schema("oregon",[
        new Field("geom","Point","EPSG:4326"),
        new Field("id","int")
])
Layer orLayer = new Memory().create(orSchema)
orLayer.withWriter { geoscript.layer.Writer writer ->
    Geometry.createRandomPoints(oregon.geom, 50).points.eachWithIndex { Point pt, int
index ->
        writer.add(orSchema.feature([geom: pt, id: index]))
    }
}
println "The Oregon Layer  has ${orLayer.count} features"

Layer mergedLayer = orLayer.merge(waLayer)
println "The merged Layer has ${mergedLayer.count} features"
```

```
The Washington Layer  has 50 features
The Oregon Layer  has 50 features
The merged Layer has 100 features
```

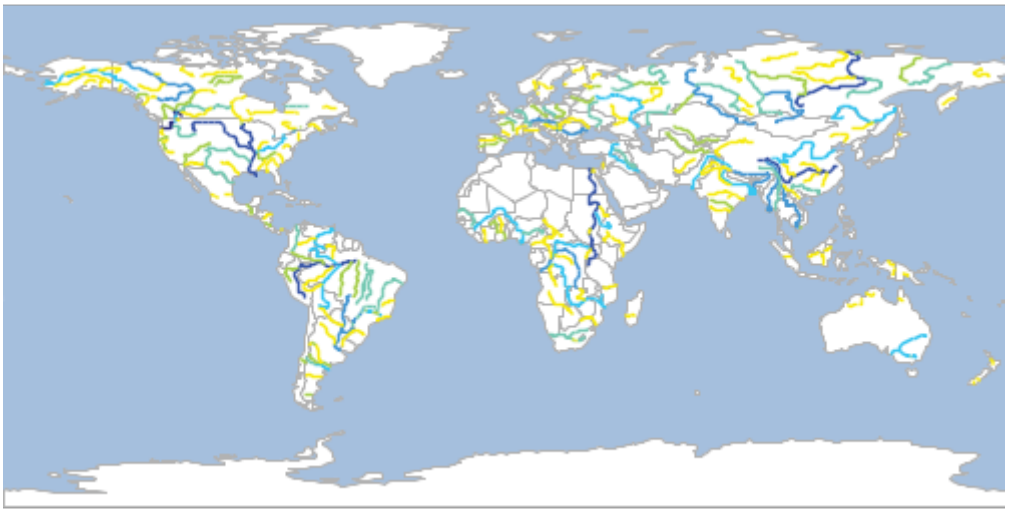


Split

Split a Layer into Layers based on the value from a Field

```
Workspace workspace = new GeoPackage(new File("src/main/resources/data.gpkg"))
Layer rivers = workspace.get("rivers")
Workspace outWorkspace = new Memory()
rivers.split(rivers.schema.get("scalerank"), outWorkspace)

outWorkspace.layers.each { Layer layer ->
    println "${layer.name} has ${layer.count} features"
}
```

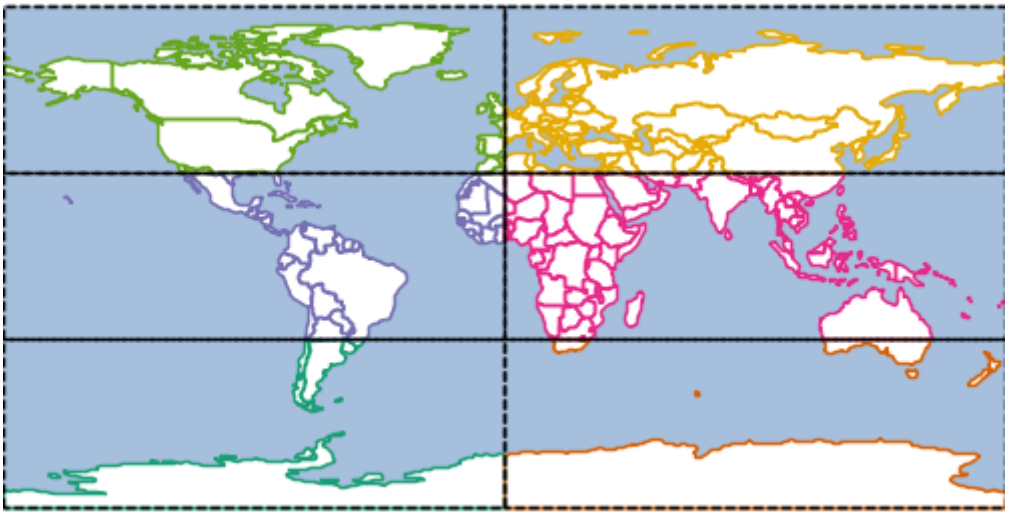
```
rivers_0 has 1 features
rivers_1 has 27 features
rivers_2 has 35 features
rivers_3 has 53 features
rivers_4 has 71 features
rivers_5 has 67 features
rivers_6 has 206 features
```



Split a Layer into Layers based on another Layer

```
Schema schema = new Schema("grid",[
        new Field("geom","Polygon","EPSG:4326"),
        new Field("col","int"),
        new Field("row","int"),
        new Field("row_col","string")
])
Workspace gridWorkspace = new Directory("target")
Layer gridLayer = gridWorkspace.create(schema)
new Bounds(-180,-90,180,90,"EPSG:4326").generateGrid(2, 3, "polygon", {cell, col, row
->
    gridLayer.add([
            "geom": cell,
            "col": col,
            "row": row,
            "row_col": "${row} ${col}"
    ])
})

Workspace workspace = new GeoPackage(new File("src/main/resources/data.gpkg"))
Layer countries = workspace.get("countries")

Workspace outWorkspace = new Memory()
countries.split(gridLayer,gridLayer.schema.get("row_col"),outWorkspace)

outWorkspace.layers.each { Layer layer ->
    println "${layer.name} has ${layer.count} features"
}
```

```
countries_1_1 has 6 features
countries_1_2 has 6 features
countries_2_1 has 44 features
countries_2_2 has 74 features
countries_3_1 has 13 features
countries_3_2 has 70 features
```

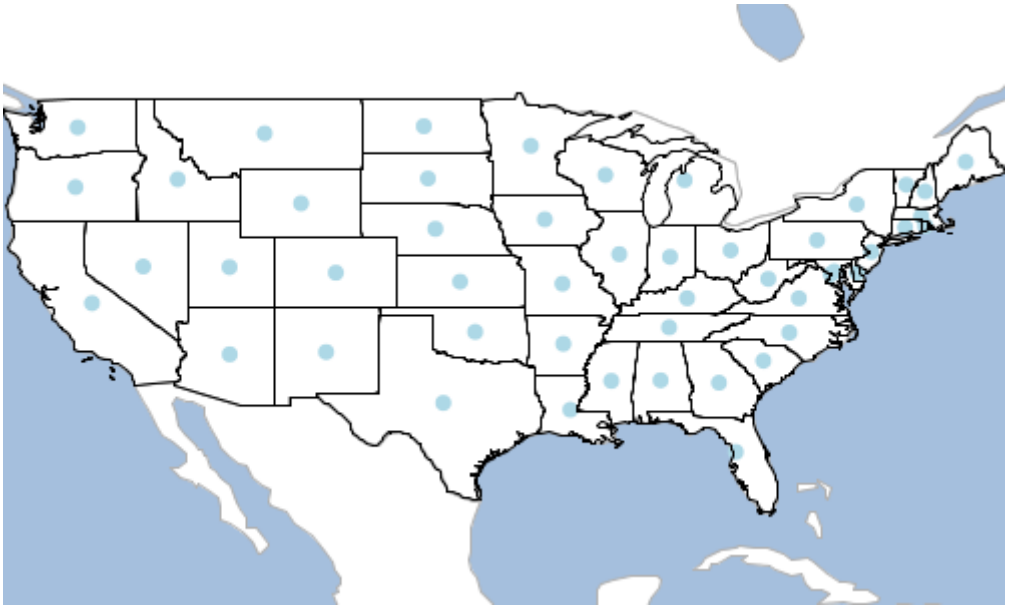


Transform

Transform one Layer into another Layer

```
Workspace workspace = new Directory(new File("src/main/resources/data"))
Layer states = workspace.get("states")
Layer centroids = states.transform("centroids", [
    geom: "centroid(the_geom)",
    name: "strToUpperCase(STATE_NAME)",
    ratio: "FEMALE / MALE"
])
centroids.eachFeature(max: 5) {Feature f ->
    println "${f.geom} ${f['name']} = ${f['ratio']}"
}
```

```
ILLINOIS = 1.0587396098110435
DISTRICT OF COLUMBIA = 1.1447503268897763
DELAWARE = 1.0626439771122835
WEST VIRGINIA = 1.0817203227723509
MARYLAND = 1.0621588832568312
```



Layer Algebra

GeoScript can do layer algebra. All of the examples below use Layer A (red) and Layer B (green).

Clip

Clip Layer A with Layer B

```
Workspace workspace = new GeoPackage(new File("src/main/resources/layeralgebra.gpkg"))
Layer layerA = workspace.get("a")
Layer layerB = workspace.get("b")
Layer layerC = layerA.clip(layerB)
```

Clip Layer B with Layer A

```
Workspace workspace = new GeoPackage(new File("src/main/resources/layeralgebra.gpkg"))
Layer layerA = workspace.get("a")
Layer layerB = workspace.get("b")
Workspace outWorkspace = new Directory("target")
Layer layerC = layerB.clip(layerA, outWorkspace: outWorkspace, outLayer: "ba_clip")
```

Erase

Erase Layer A with Layer B

```
Workspace workspace = new GeoPackage(new File("src/main/resources/layeralgebra.gpkg"))
Layer layerA = workspace.get("a")
Layer layerB = workspace.get("b")
Layer layerC = layerA.erase(layerB)
```

Erase Layer B with Layer A

```
Workspace workspace = new GeoPackage(new File("src/main/resources/layeralgebra.gpkg"))
Layer layerA = workspace.get("a")
Layer layerB = workspace.get("b")
Workspace outWorkspace = new Directory("target")
Layer layerC = layerB.erase(layerA, outWorkspace: outWorkspace, outLayer: "ba_erase")
```

Identity

Identity Layer A with Layer B

```
Workspace workspace = new GeoPackage(new File("src/main/resources/layeralgebra.gpkg"))
Layer layerA = workspace.get("a")
Layer layerB = workspace.get("b")
Layer layerC = layerA.identity(layerB)
```

Identity Layer B with Layer A

```
Workspace workspace = new GeoPackage(new File("src/main/resources/layeralgebra.gpkg"))
Layer layerA = workspace.get("a")
Layer layerB = workspace.get("b")
Workspace outWorkspace = new Directory("target")
Layer layerC = layerB.identity(layerA, outWorkspace: outWorkspace, outLayer:
"ba_identity")
```

Intersection

Intersection Layer A with Layer B

```
Workspace workspace = new GeoPackage(new File("src/main/resources/layeralgebra.gpkg"))
Layer layerA = workspace.get("a")
Layer layerB = workspace.get("b")
Layer layerC = layerA.intersection(layerB)
```

Intersection Layer B with Layer A

```
Workspace workspace = new GeoPackage(new File("src/main/resources/layeralgebra.gpkg"))
Layer layerA = workspace.get("a")
Layer layerB = workspace.get("b")
Workspace outWorkspace = new Directory("target")
Layer layerC = layerB.intersection(layerA, outWorkspace: outWorkspace, outLayer:
"ba_intersection")
```

Symmetric Difference

Symmetric Difference Layer A with Layer B

```
Workspace workspace = new GeoPackage(new File("src/main/resources/layeralgebra.gpkg"))
Layer layerA = workspace.get("a")
Layer layerB = workspace.get("b")
Layer layerC = layerA.symDifference(layerB)
```

Symmetric Difference Layer B with Layer A

```
Workspace workspace = new GeoPackage(new File("src/main/resources/layeralgebra.gpkg"))
Layer layerA = workspace.get("a")
Layer layerB = workspace.get("b")
Workspace outWorkspace = new Directory("target")
Layer layerC = layerB.symDifference(layerA, outWorkspace: outWorkspace, outLayer:
"ba_symdifference")
```

Update

Update Layer A with Layer B

```
Workspace workspace = new GeoPackage(new File("src/main/resources/layeralgebra.gpkg"))
Layer layerA = workspace.get("a")
Layer layerB = workspace.get("b")
Layer layerC = layerA.update(layerB)
```

Update Layer B with Layer A

```
Workspace workspace = new GeoPackage(new File("src/main/resources/layeralgebra.gpkg"))
Layer layerA = workspace.get("a")
Layer layerB = workspace.get("b")
Workspace outWorkspace = new Directory("target")
Layer layerC = layerB.update(layerA, outWorkspace: outWorkspace, outLayer:
"ba_update")
```

Union

Union Layer A with Layer B

```
Workspace workspace = new GeoPackage(new File("src/main/resources/layeralgebra.gpkg"))
Layer layerA = workspace.get("a")
Layer layerB = workspace.get("b")
Layer layerC = layerA.union(layerB)
```

Union Layer B with Layer A

```
Workspace workspace = new GeoPackage(new File("src/main/resources/layeralgebra.gpkg"))
Layer layerA = workspace.get("a")
Layer layerB = workspace.get("b")
Workspace outWorkspace = new Directory("target")
Layer layerC = layerB.union(layerA, outWorkspace: outWorkspace, outLayer: "ba_union")
```

Reading and Writing Features

The Layer IO classes are in the **geoscript.layer.io** package.

Finding Layer Writer and Readers

List all Layer Writers

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

```
CsvWriter
GeobufWriter
GeoJSONWriter
GeoRSSWriter
GmlWriter
GpxWriter
KmlWriter
MvtWriter
```

Find a Layer Writer

```
Workspace workspace = new Memory()
Schema schema = new Schema("cities", [
        new Field("geom", "Point", "EPSG:4326"),
        new Field("id", "Integer"),
        new Field("name", "String")
])
Layer layer = workspace.create(schema)
layer.add([
        geom: new Point(-122.3204, 47.6024),
        id: 1,
        name: "Seattle"
])
layer.add([
        geom: new Point(-122.48416, 47.2619),
        id: 2,
        name: "Tacoma"
])

Writer writer = Writers.find("csv")
String csv = writer.write(layer)
println csv
```

```
"geom:Point:EPSG:4326","id:Integer","name:String"
"POINT (-122.3204 47.6024)","1","Seattle"
"POINT (-122.48416 47.2619)","2","Tacoma"
```

List all Layer Readers

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

```
CsvReader
GeobufReader
GeoJSONReader
GeoRSSReader
GmlReader
GpxReader
KmlReader
MvtReader
```

Find a Layer Reader

```
        Reader reader = Readers.find("csv")
        Layer layer = reader.read("""geom:Point:EPSG:4326","id:Integer","name:String"
"POINT (-122.3204 47.6024)","1","Seattle"
"POINT (-122.48416 47.2619)","2","Tacoma"
""")
        println "# features = ${layer.count}"
```

```
# features = 2
```

GeoJSON

Get GeoJSON String from a Layer

```
Workspace workspace = new Memory()
Schema schema = new Schema("cities", [
        new Field("geom", "Point", "EPSG:4326"),
        new Field("id", "Integer"),
        new Field("name", "String")
])
Layer layer = workspace.create(schema)
layer.add([
    geom: new Point(-122.3204, 47.6024),
    id: 1,
    name: "Seattle"
])
layer.add([
    geom: new Point(-122.48416, 47.2619),
    id: 2,
    name: "Tacoma"
])

String geojson = layer.toJSONString()
println geojson
```

```
{
    "type": "FeatureCollection",
    "features": [
        {
            "type": "Feature",
            "geometry": {
                "type": "Point",
                "coordinates": [
                    -122.3204,
                    47.6024
                ]
            },
            "properties": {
                "id": 1,
                "name": "Seattle"
            },
            "id": "fid-5b6b299_1620e77d8c3_-6adf"
        },
        {
            "type": "Feature",
            "geometry": {
                "type": "Point",
                "coordinates": [
                    -122.4842,
                    47.2619
                ]
            },
            "properties": {
                "id": 2,
                "name": "Tacoma"
            },
            "id": "fid-5b6b299_1620e77d8c3_-6add"
        }
    ]
}
```


GeoBuf

Get GeoBuf String from a Layer

```
Workspace workspace = new Memory()
Schema schema = new Schema("cities", [
        new Field("geom", "Point", "EPSG:4326"),
        new Field("id", "Integer"),
        new Field("name", "String")
])
Layer layer = workspace.create(schema)
layer.add([
        geom: new Point(-122.3204, 47.6024),
        id: 1,
        name: "Seattle"
])
layer.add([
        geom: new Point(-122.48416, 47.2619),
        id: 2,
        name: "Tacoma"
])

String geobuf = layer.toGeobufString()
println geobuf
```

```
0a0269640a046e616d6510021806223d0a1d0a0c08001a089fd8d374c0ebb22d6a0218016a090a07536561
74746c650a1c0a0c08001a08ffd6e77498a3892d6a0218026a080a065461636f6d61
```

GML

Get GML String from a Layer

```
Workspace workspace = new Memory()
Schema schema = new Schema("cities", [
        new Field("geom", "Point", "EPSG:4326"),
        new Field("id", "Integer"),
        new Field("name", "String")
])
Layer layer = workspace.create(schema)
layer.add([
        geom: new Point(-122.3204, 47.6024),
        id: 1,
        name: "Seattle"
])
layer.add([
        geom: new Point(-122.48416, 47.2619),
        id: 2,
        name: "Tacoma"
])

String gml = layer.toGMLString()
println gml
```

```
<wfs:FeatureCollection xmlns:wfs="http://www.opengis.net/wfs">
  <gml:boundedBy xmlns:gml="http://www.opengis.net/gml">
    <gml:Box srsName="http://www.opengis.net/gml/srs/epsg.xml#4326">
      <gml:coord>
        <gml:X>
          -122.48416
        </gml:X>
        <gml:Y>
          47.2619
        </gml:Y>
      </gml:coord>
      <gml:coord>
        <gml:X>
          -122.3204
        </gml:X>
        <gml:Y>
          47.6024
        </gml:Y>
      </gml:coord>
    </gml:Box>
  </gml:boundedBy>
  <gml:featureMember xmlns:gml="http://www.opengis.net/gml">
    <gsf:cities xmlns:gsf="http://geoscript.org/feature" fid="fid-
5b6b299_1620e77d8c3_-7491">
      <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>
  </gml:featureMember>
  <gml:featureMember xmlns:gml="http://www.opengis.net/gml">
    <gsf:cities xmlns:gsf="http://geoscript.org/feature" fid="fid-
5b6b299_1620e77d8c3_-748f">
      <gml:name>
        Tacoma
      </gml:name>
      <gsf:geom>
        <gml:Point>
          <gml:coord>
            <gml:X>
              -122.48416
            </gml:X>
            <gml:Y>
              47.2619
            </gml:Y>
          </gml:coord>
        </gml:Point>
      </gsf:geom>
      <gsf:id>
        2
      </gsf:id>
    </gsf:cities>
  </gml:featureMember>
</wfs:FeatureCollection>
```

KML

Get KML String from a Layer

```
Workspace workspace = new Memory()
Schema schema = new Schema("cities", [
        new Field("geom", "Point", "EPSG:4326"),
        new Field("id", "Integer"),
        new Field("name", "String")
])
Layer layer = workspace.create(schema)
layer.add([
    geom: new Point(-122.3204, 47.6024),
    id: 1,
    name: "Seattle"
])
layer.add([
    geom: new Point(-122.48416, 47.2619),
    id: 2,
    name: "Tacoma"
])

String kml = layer.toKMLString()
println kml
```

```
<kml:kml xmlns:kml="http://www.opengis.net/kml/2.2">
  <kml:Document>
    <kml:Folder>
      <kml:name>
        cities
      </kml:name>
      <kml:Schema kml:name="cities" kml:id="cities">
        <kml:SimpleField kml:name="id" kml:type="Integer"/>
        <kml:SimpleField kml:name="name" kml:type="String"/>
      </kml:Schema>
      <kml:Placemark>
        <kml:name>
          fid-5b6b299_1620e77d8c3_-6b14
        </kml:name>
        <kml:Style>
          <kml:IconStyle>
            <kml:color>
              ff0000ff
            </kml:color>
          </kml:IconStyle>
        </kml:Style>
        <kml:ExtendedData>
          <kml:SchemaData kml:schemaUrl="#cities">
            <kml:SimpleData kml:name="id">
              1
            </kml:SimpleData>
            <kml:SimpleData kml:name="name">
```

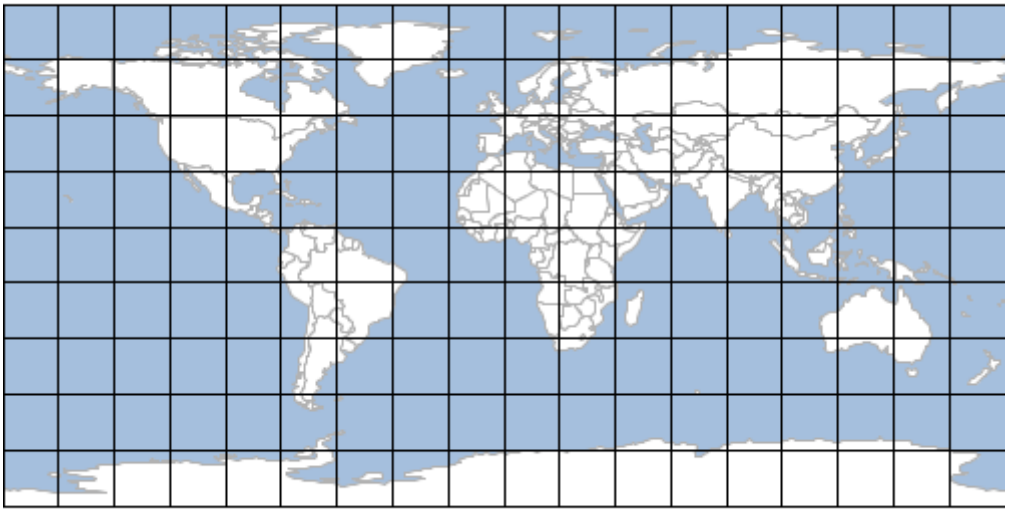
```
              Seattle
            </kml:SimpleData>
          </kml:SchemaData>
        </kml:ExtendedData>
        <kml:Point>
          <kml:coordinates>
            -122.3204,47.6024
          </kml:coordinates>
        </kml:Point>
      </kml:Placemark>
      <kml:Placemark>
        <kml:name>
          fid-5b6b299_1620e77d8c3_-6b12
        </kml:name>
        <kml:Style>
          <kml:IconStyle>
            <kml:color>
              ff0000ff
            </kml:color>
          </kml:IconStyle>
        </kml:Style>
        <kml:ExtendedData>
          <kml:SchemaData kml:schemaUrl="#cities">
            <kml:SimpleData kml:name="id">
              2
            </kml:SimpleData>
            <kml:SimpleData kml:name="name">
              Tacoma
            </kml:SimpleData>
          </kml:SchemaData>
        </kml:ExtendedData>
        <kml:Point>
          <kml:coordinates>
            -122.48416,47.2619
          </kml:coordinates>
        </kml:Point>
      </kml:Placemark>
    </kml:Folder>
  </kml:Document>
</kml:kml>
```

Graticules

Square

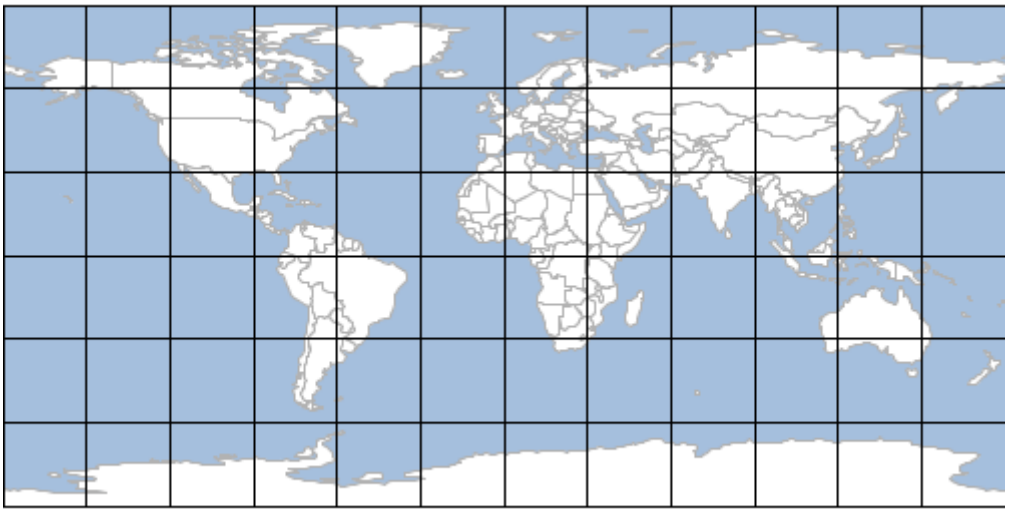
Create a square graticules Layer

```
Bounds bounds = new Bounds(-180,-90,180,90,"EPSG:4326")
double length = 20
double spacing = 5
Layer layer = Graticule.createSquares(bounds, length, spacing)
```



Create a square graticules Shapefile Layer

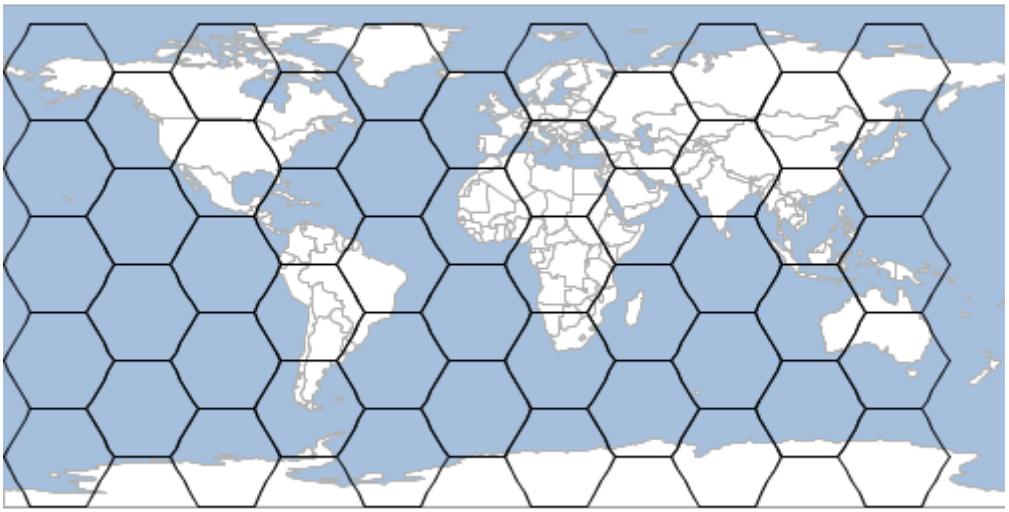
```
Bounds bounds = new Bounds(-180,-90,180,90,"EPSG:4326")
double length = 30
double spacing = -1
Workspace workspace = new Directory("target")
Layer layer = Graticule.createSquares(bounds, length, spacing, workspace: workspace,
layer: "squares")
```



Hexagon

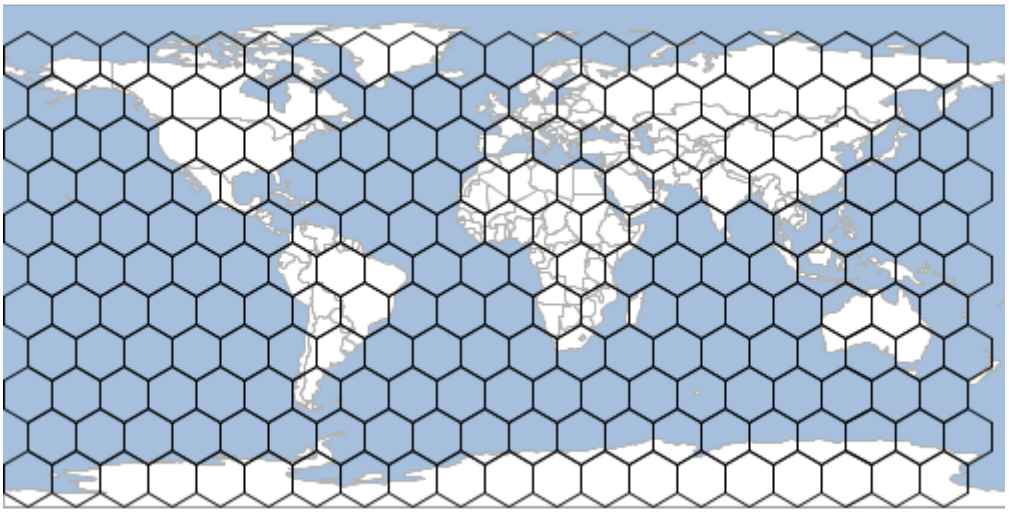
Create a flat hexagon graticules Layer

```
Bounds bounds = new Bounds(-180,-90,180,90,"EPSG:4326")
double length = 20
double spacing = 5
String orientation = "flat"
Layer layer = Graticule.createHexagons(bounds, length, spacing, orientation)
```



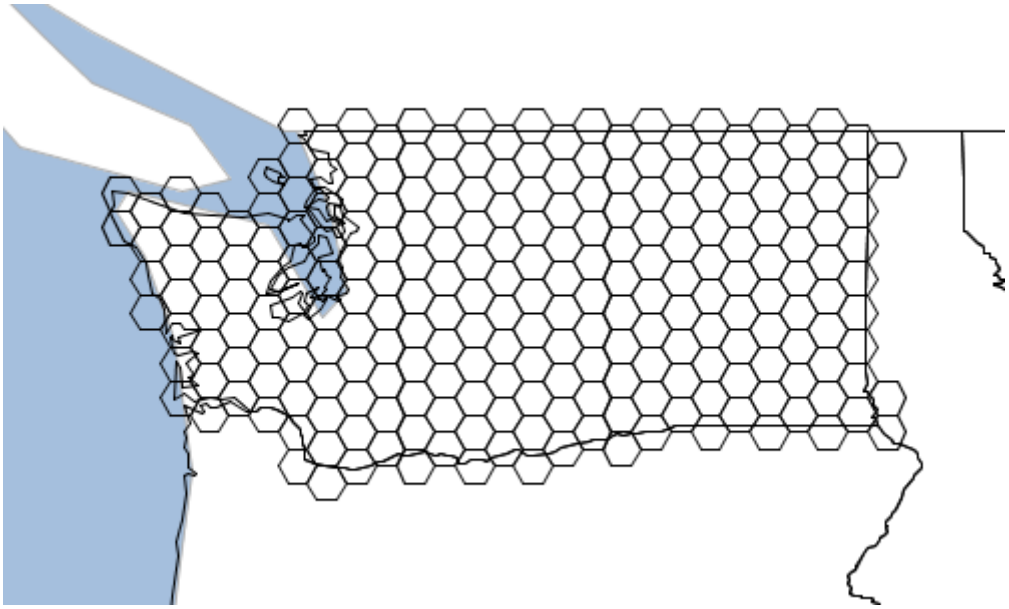
Create a angled hexagon graticules Layer

```
Bounds bounds = new Bounds(-180,-90,180,90,"EPSG:4326")
double length = 10
double spacing = 5
String orientation = "angled"
Layer layer = Graticule.createHexagons(bounds, length, spacing, orientation)
```



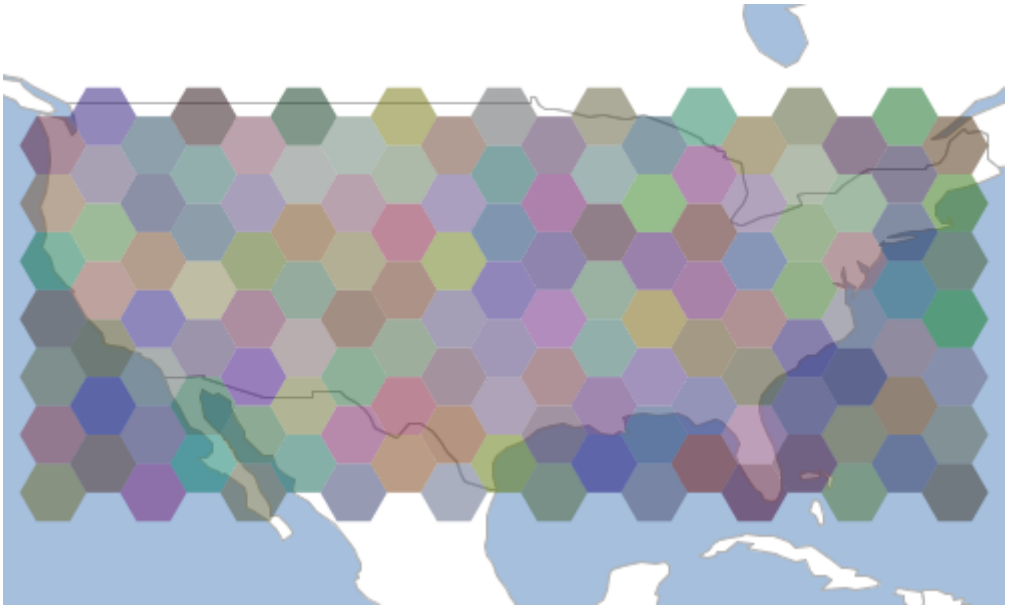
Create a hexagon graticules Layer intersecting Washington States

```
Layer states = new Shapefile("src/main/resources/data/states.shp")
Feature feature = states.first(filter: "STATE_NAME = 'Washington'")
Layer layer = Graticule.createHexagons(feature.bounds.expandBy(1.0), 0.2, -1.0,
"flat", createFeature: { GridElement e ->
    new Point(e.center.x, e.center.y).buffer(0.2).intersects(feature.geom)
})
```



Create a hexagon graticules Layer with a custom schema

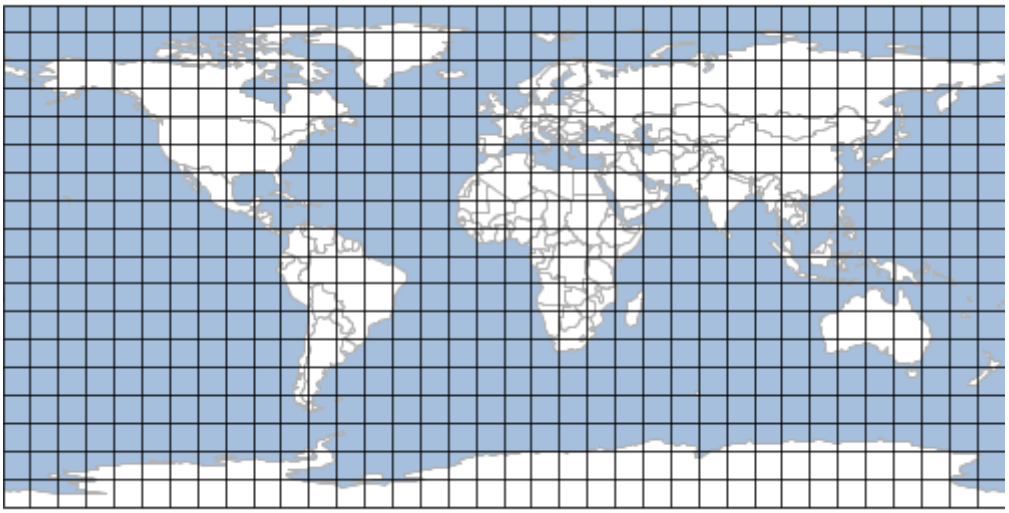
```
Layer states = new Shapefile("src/main/resources/data/states.shp")
Schema schema = new Schema("hexagon", [
        new Field("geom", "Polygon"),
        new Field("color", "java.awt.Color")
])
Bounds b = states.bounds.expandBy(1)
Layer layer = Graticule.createHexagons(b, 2.0, -1.0, "flat", schema: schema,
setAttributes: { GridElement e, Map attributes ->
    attributes["color"] = Color.randomPastel.asColor()
})
layer.style = new Fill(new Property("color"), 0.5)
```



Line

Create a line graticules Layer

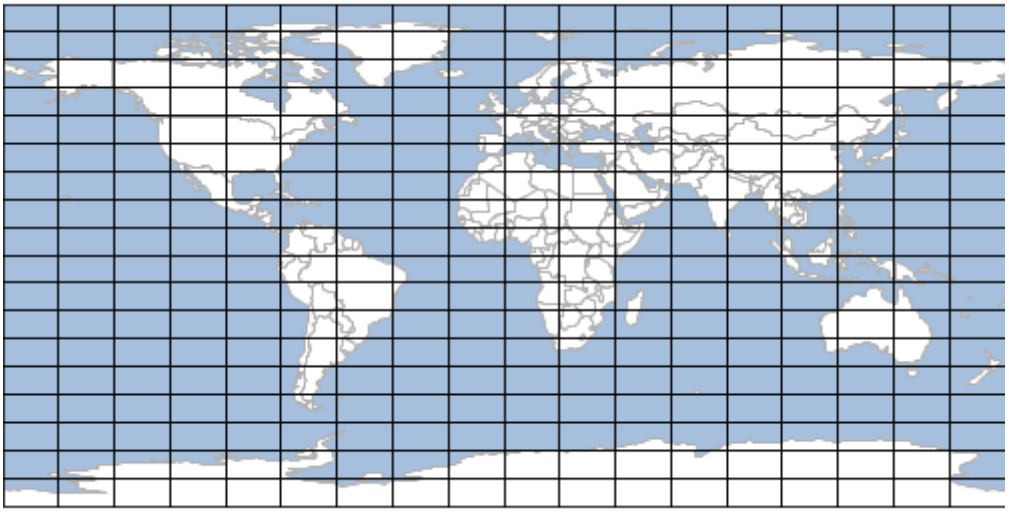
```
Bounds bounds = new Bounds(-180,-90,180,90,"EPSG:4326")
Layer layer = Graticule.createLines(bounds, [
        [orientation: 'vertical',   level: 2, spacing: 20],
        [orientation: 'vertical',   level: 1, spacing: 10 ],
        [orientation: 'horizontal', level: 2, spacing: 20],
        [orientation: 'horizontal', level: 1, spacing: 10 ]
], 2.0)
```



Rectangle

Create a rectangular graticules Layer

```
Bounds bounds = new Bounds(-180,-90,180,90,"EPSG:4326")
double width = 20
double height = 10
Layer layer = Graticule.createRectangles(bounds, width, height, -1)
```



Format Recipes

The Format classes are in the **geoscript.layer** package.

A Format is a collection of Rasters.

Get a Format

Get a Format from a File

```
File file = new File("src/main/resources/earth.tif")
Format format = Format.getFormat(file)
println format.name
```

```
GeoTIFF
```

Get Names

Get names of the Rasters in a Format. Some Formats can contain more than one Raster.

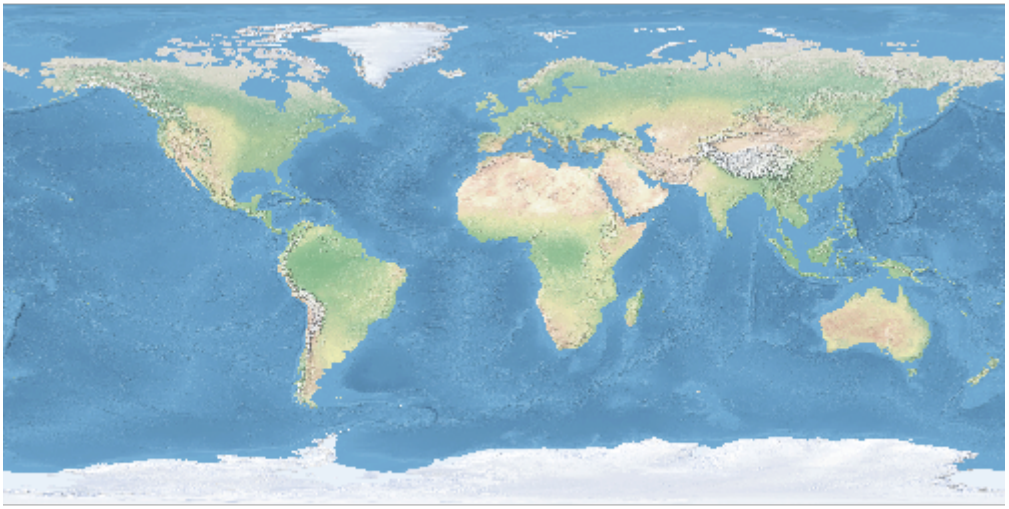
```
File file = new File("src/main/resources/earth.tif")
Format format = Format.getFormat(file)
List<String> names = format.names
names.each { String name ->
    println name
}
```

```
earth
```

Read a Raster

Read a Raster from a File

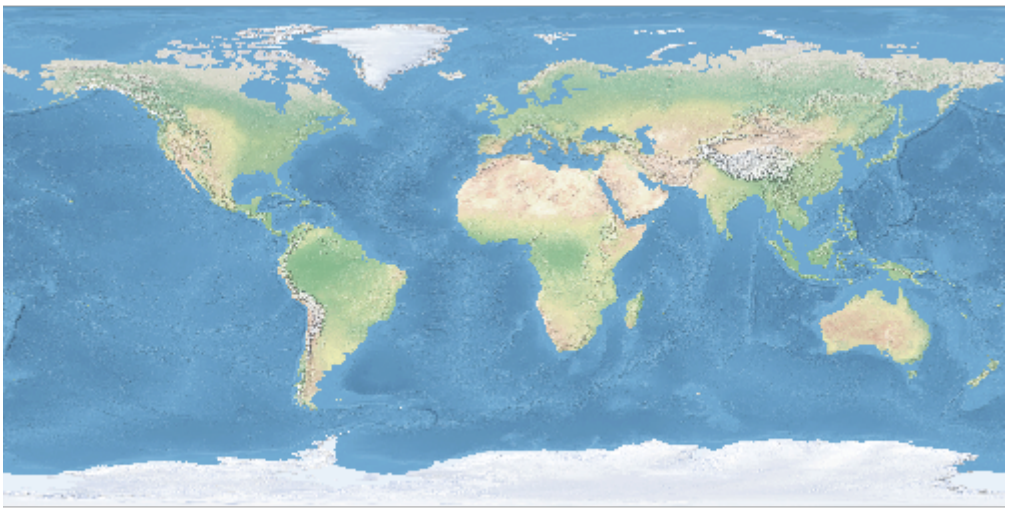
```
File file = new File("src/main/resources/earth.tif")
Format format = Format.getFormat(file)
Raster raster = format.read("earth")
```



Write a Raster

Write a Raster to a File

```
File file = new File("src/main/resources/earth.tif")
Format format = Format.getFormat(file)
Raster raster = format.read("earth")

File outFile = new File("target/earth.png")
Format outFormat = Format.getFormat(outFile)
outFormat.write(raster)
Raster outRaster = outFormat.read("earth")
```



Check for a Raster

Check to see if the Format has a Raster

```
File file = new File("src/main/resources/earth.tif")
Format format = Format.getFormat(file)

boolean hasEarth = format.has("earth")
println "Has raster named earth? ${hasEarth}"

boolean hasWorld = format.has("world")
println "Has raster named world? ${hasWorld}"
```

```
Has raster named earth? true
Has raster named world? false
```

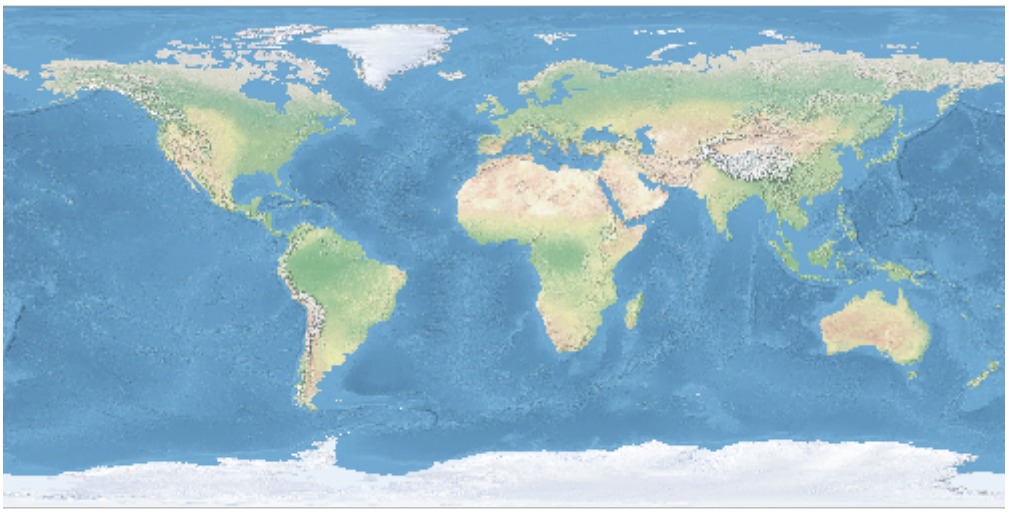
Raster Recipes

The Raster classes are in the **geoscript.layer** package.

Raster Properties

Read a Raster from a File

```
File file = new File("src/main/resources/earth.tif")
Format format = Format.getFormat(file)
Raster raster = format.read("earth")
```



Get the Raster's Bounds.

```
Bounds bounds = raster.bounds
println "Bounds: ${bounds}"
```

```
Bounds: (-179.99999999999997,-89.99999999998205,179.99999999996405,90.0,EPSG:4326)
```

Get the Raster's Projection.

```
Projection projection = raster.proj
println "Projection: ${projection}"
```

```
Projection: EPSG:4326
```

Get the Raster's Size.

```
List size = raster.size
println "Size: ${size[0]}x${size[1]}"
```

```
Size: 800x400
```

Get the Raster's number of columns and rows.

```
int cols = raster.cols
int rows = raster.rows
println "Columns: ${cols} Rows: ${rows}"
```

```
Columns: 800 Rows: 400
```

Get the Raster's Bands.

```
List<Band> bands = raster.bands
println "Bands:"
bands.each { Band band ->
    println "  ${band}"
}
```

```
Band:
  RED_BAND
  GREEN_BAND
  BLUE_BAND
```

Get the Raster's block size.

```
List blockSize = raster.blockSize
println "Block size: ${blockSize[0]}x${blockSize[1]}"
```

```
Block size: 800x8
```

Get the Raster's pixel size.

```
List pixelSize = raster.pixelSize
println "Pixel size: ${pixelSize[0]}x${pixelSize[1]}"
```

```
Pixel size: 0.44999999999995505x0.4499999999999551
```

Get more information about a Raster's Bounds.

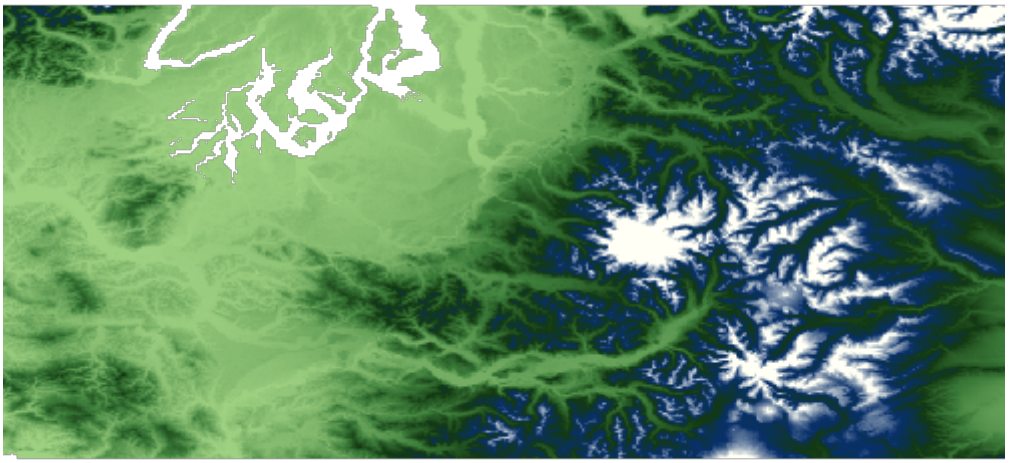
```
File file = new File("src/main/resources/earth.tif")
Format format = Format.getFormat(file)
Raster raster = format.read("earth")
List<Band> bands = raster.bands
bands.each { Band band ->
    println "${band}"
    println "  Min = ${band.min}"
    println "  Max = ${band.max}"
    println "  No Data = ${band.noData}"
    println "  Is No Data = ${band.isNoData(12.45)}"
    println "  Unit = ${band.unit}"
    println "  Scale = ${band.scale}"
    println "  Offset = ${band.offset}"
    println "  Type = ${band.type}"
}
```

```
RED_BAND
  Min = 0.0
  Max = 0.0
  No Data = [0.0]
  Is No Data = false
  Unit = null
  Scale = 1.0
  Offset = 0.0
  Type = byte

GREEN_BAND
  Min = 0.0
  Max = 0.0
  No Data = [0.0]
  Is No Data = false
  Unit = null
  Scale = 1.0
  Offset = 0.0
  Type = byte

BLUE_BAND
  Min = 0.0
  Max = 0.0
  No Data = [0.0]
  Is No Data = false
  Unit = null
  Scale = 1.0
  Offset = 0.0
  Type = byte
```

Raster Values

Get values from a Raster

```
File file = new File("src/main/resources/pc.tif")
Format format = Format.getFormat(file)
Raster raster = format.read("pc")
```



Get values from a Raster with a Point.

```
double elevation = raster.getValue(new Point(-121.799927,46.867703))
println elevation
```

```
3069.0
```

Get values from a Raster with a Pixel Location.

```
List pixel = [100,200]
elevation = raster.getValue(pixel)
println elevation
```

```
288.0
```

Raster Processing

Crop

Crop a Raster with a Bounds

```
File file = new File("src/main/resources/earth.tif")
Format format = Format.getFormat(file)
Raster raster = format.read("earth")
Raster croppedRaster = raster.crop(new Bounds(-160.927734,6.751896,-34.716797
,57.279043, "EPSG:4326"))
```

Project

Reproject a Raster to another Projection

```
File file = new File("src/main/resources/earth.tif")
Format format = Format.getFormat(file)
Raster raster = format.read("earth")
Projection projection = new Projection("EPSG:3857")
Raster projectedRaster = raster.crop(projection.geoBounds).reproject(projection)
```

Tile Recipes

The Tile classes are in the **geoscript.layer** package.

Tile

Tile Properties

Get a Tile's Properties.

```
byte[] data = new File("src/main/resources/tile.png").bytes
Tile tile = new Tile(2,1,3,data)
println "Z = ${tile.z}"
println "X = ${tile.x}"
println "Y = ${tile.y}"
println "Tile = ${tile.toString()}"
println "# bytes = ${tile.data.length}"
println "Data as base64 encoded string = ${tile.base64String}"
```

```
Z = 2
X = 1
Y = 3
Tile = Tile(x:1, y:3, z:2)
# bytes = 11738
Data as base64 encoded string = iVBORw0KGgoAAAANSUhEUgAAAQAAAAEACAYAAABccqhmAAAtoU...
```

ImageTile Properties

Some Tiles contain an Image. ImageTile's have an image property.

```
byte[] data = new File("src/main/resources/tile.png").bytes
ImageTile tile = new ImageTile(0,0,0,data)
BufferedImage image = tile.image
```

Grid

A Grid describes a level in a Pyramid of Tiles.

Grid Properties

```
Grid grid = new Grid(1, 2, 2, 78206.0, 78206.0)
println "Zoom Level: ${grid.z}"
println "Width / # Columns: ${grid.width}"
println "Height / # Rows: ${grid.height}"
println "Size / # Tiles: ${grid.size}"
println "X Resolution: ${grid.xResolution}"
println "Y Resolution: ${grid.yResolution}"
```

```
Zoom Level: 1
Width / # Columns: 2
Height / # Rows: 2
Size / # Tiles: 4
X Resolution: 78206.0
Y Resolution: 78206.0
```

Pyramid

Pyramid Properties

Get the Pyramid's Bounds.

```
Pyramid pyramid = Pyramid.createGlobalMercatorPyramid()

Bounds bounds = pyramid.bounds
println bounds
```

```
(-2.0036395147881314E7,-
2.0037471205137067E7,2.0036395147881314E7,2.003747120513706E7,EPSG:3857)
```

Get the Pyramid's projection.

```
Projection proj = pyramid.proj
println proj
```

```
EPSG:3857
```

Get the Pyramid's Origin.

```
Pyramid.Origin origin = pyramid.origin
println origin
```

```
BOTTOM_LEFT
```

Get the Pyramid's Tile Width and Height.

```
int tileWidth = pyramid.tileWidth
int tileHeight = pyramid.tileHeight
println "${tileWidth} x ${tileHeight}"
```

```
256 x 256
```

Create Pyramids

Create a Global Mercator Pyramid.

```
Pyramid pyramid = Pyramid.createGlobalMercatorPyramid()
println "Projection: ${pyramid.proj}"
println "Origin: ${pyramid.origin}"
println "Bounds: ${pyramid.bounds}"
println "Max Zoom: ${pyramid.maxGrid.z}"
```

```
Projection: EPSG:3857
Origin: BOTTOM_LEFT
Bounds: (-2.0036395147881314E7,-
2.0037471205137067E7,2.0036395147881314E7,2.003747120513706E7,EPSG:3857)
Max Zoom: 19
```

Create a Global Geodetic Pyramid.

```
Pyramid pyramid = Pyramid.createGlobalGeodeticPyramid()
println "Projection: ${pyramid.proj}"
println "Origin: ${pyramid.origin}"
println "Bounds: ${pyramid.bounds}"
println "Max Zoom: ${pyramid.maxGrid.z}"
```

```
Projection: EPSG:4326
Origin: BOTTOM_LEFT
Bounds: (-179.99,-89.99,179.99,89.99,EPSG:4326)
Max Zoom: 19
```

Create a Global Mercator Pyramid from a well known name.

Well known names include:

- GlobalMercator
- Mercator
- GlobalMercatorBottomLeft
- GlobalMercatorTopLeft
- GlobalGeodetic
- Geodetic

```
Pyramid pyramid = Pyramid.fromString("mercator")
println "Projection: ${pyramid.proj}"
println "Origin: ${pyramid.origin}"
println "Bounds: ${pyramid.bounds}"
println "Max Zoom: ${pyramid.maxGrid.z}"
```

```
Projection: EPSG:3857
Origin: BOTTOM_LEFT
Bounds: (-2.0036395147881314E7,-
2.0037471205137067E7,2.0036395147881314E7,2.003747120513706E7,EPSG:3857)
Max Zoom: 19
```

Get a Grid from a Pyramid

Get a the min Grid.

```
Pyramid pyramid = Pyramid.createGlobalMercatorPyramid()
Grid grid = pyramid.minGrid
println "Zoom Level: ${grid.z}"
println "Width / # Columns: ${grid.width}"
println "Height / # Rows: ${grid.height}"
println "Size / # Tiles: ${grid.size}"
println "X Resolution: ${grid.xResolution}"
println "Y Resolution: ${grid.yResolution}"
```

```
Zoom Level: 0
Width / # Columns: 1
Height / # Rows: 1
Size / # Tiles: 1
X Resolution: 156412.0
Y Resolution: 156412.0
```

Get a the max Grid.

```
Pyramid pyramid = Pyramid.createGlobalMercatorPyramid()
Grid grid = pyramid.maxGrid
println "Zoom Level: ${grid.z}"
println "Width / # Columns: ${grid.width}"
println "Height / # Rows: ${grid.height}"
println "Size / # Tiles: ${grid.size}"
println "X Resolution: ${grid.xResolution}"
println "Y Resolution: ${grid.yResolution}"
```

```
Zoom Level: 19
Width / # Columns: 524288
Height / # Rows: 524288
Size / # Tiles: 274877906944
X Resolution: 0.29833221435546875
Y Resolution: 0.29833221435546875
```

Get a Grid from a Pyramid by Zoom Level.

```
Pyramid pyramid = Pyramid.createGlobalMercatorPyramid()
Grid grid = pyramid.grid(1)
println "Zoom Level: ${grid.z}"
println "Width / # Columns: ${grid.width}"
println "Height / # Rows: ${grid.height}"
println "Size / # Tiles: ${grid.size}"
println "X Resolution: ${grid.xResolution}"
println "Y Resolution: ${grid.yResolution}"
```

```
Zoom Level: 1
Width / # Columns: 2
Height / # Rows: 2
Size / # Tiles: 4
X Resolution: 78206.0
Y Resolution: 78206.0
```

Reading and Writing Pyramids

The Pyramid IO classes are in the **geoscript.layer.io** package.

Finding Pyramid Writer and Readers

List all Pyramid Writers

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

```
CsvPyramidWriter
GdalTmsPyramidWriter
JsonPyramidWriter
XmlPyramidWriter
```

Find a Pyramid Writer

```
Pyramid pyramid = Pyramid.createGlobalGeodeticPyramid(maxZoom: 2)
PyramidWriter writer = PyramidWriters.find("csv")
String pyramidStr = writer.write(pyramid)
println pyramidStr
```

```
EPSG:4326
-179.99,-89.99,179.99,89.99,EPSG:4326
BOTTOM_LEFT
256,256
0,2,1,0.703125,0.703125
1,4,2,0.3515625,0.3515625
2,8,4,0.17578125,0.17578125
```

List all Pyramid Readers

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

```
CsvPyramidReader
GdalTmsPyramidReader
JsonPyramidReader
XmlPyramidReader
```

Find a Pyramid Reader

```
        PyramidReader reader = PyramidReaders.find("csv")
        Pyramid pyramid = reader.read("""EPSG:3857
-2.0036395147881314E7,
-2.0037471205137067E7,2.0036395147881314E7,2.003747120513706E7,EPSG:3857
BOTTOM_LEFT
256,256
0,1,1,156412.0,156412.0
1,2,2,78206.0,78206.0
2,4,4,39103.0,39103.0
3,8,8,19551.5,19551.5
4,16,16,9775.75,9775.75
""")
        println pyramid
```

```
geoscript.layer.Pyramid(proj:EPSG:3857, bounds:(-2.0036395147881314E7,-
2.0037471205137067E7,2.0036395147881314E7,2.003747120513706E7,EPSG:3857),
origin:BOTTOM_LEFT, tileWidth:256, tileHeight:256)
```

JSON

Get a JSON String from a Pyramid.

```
Pyramid pyramid = Pyramid.createGlobalMercatorPyramid(maxZoom: 4)
String json = pyramid.json
println json
```


```
{
    "proj": "EPSG:3857",
    "bounds": {
        "minX": -2.0036395147881314E7,
        "minY": -2.0037471205137067E7,
        "maxX": 2.0036395147881314E7,
        "maxY": 2.003747120513706E7
    },
    "origin": "BOTTOM_LEFT",
    "tileSize": {
        "width": 256,
        "height": 256
    },
    "grids": [
        {
            "z": 0,
            "width": 1,
            "height": 1,
            "xres": 156412.0,
            "yres": 156412.0
        },
        {
            "z": 1,
            "width": 2,
            "height": 2,
            "xres": 78206.0,
            "yres": 78206.0
        },
        {
            "z": 2,
            "width": 4,
            "height": 4,
            "xres": 39103.0,
            "yres": 39103.0
        },
        {
            "z": 3,
            "width": 8,
            "height": 8,
            "xres": 19551.5,
            "yres": 19551.5
        },
        {
            "z": 4,
            "width": 16,
```

```
            "height": 16,
            "xres": 9775.75,
            "yres": 9775.75
        }
    ]
}
```

XML

Get a XML String from a Pyramid.

```
Pyramid pyramid = Pyramid.createGlobalMercatorPyramid(maxZoom: 4)
String xml = pyramid.xml
println xml
```

```
<pyramid>
  <proj>EPSG:3857</proj>
  <bounds>
    <minX>-2.0036395147881314E7</minX>
    <minY>-2.0037471205137067E7</minY>
    <maxX>2.0036395147881314E7</maxX>
    <maxY>2.003747120513706E7</maxY>
  </bounds>
  <origin>BOTTOM_LEFT</origin>
  <tileSize>
    <width>256</width>
    <height>256</height>
  </tileSize>
  <grids>
    <grid>
      <z>0</z>
      <width>1</width>
      <height>1</height>
      <xres>156412.0</xres>
      <yres>156412.0</yres>
    </grid>
    <grid>
      <z>1</z>
      <width>2</width>
      <height>2</height>
      <xres>78206.0</xres>
      <yres>78206.0</yres>
    </grid>
    <grid>
      <z>2</z>
      <width>4</width>
      <height>4</height>
      <xres>39103.0</xres>
      <yres>39103.0</yres>
```

```
    </grid>
    <grid>
      <z>3</z>
      <width>8</width>
      <height>8</height>
      <xres>19551.5</xres>
      <yres>19551.5</yres>
    </grid>
    <grid>
      <z>4</z>
      <width>16</width>
      <height>16</height>
      <xres>9775.75</xres>
      <yres>9775.75</yres>
    </grid>
  </grids>
</pyramid>
```

CSV

Get a CSV String from a Pyramid.

```
Pyramid pyramid = Pyramid.createGlobalMercatorPyramid(maxZoom: 4)
String csv = pyramid.csv
println csv
```

```
EPSG:3857
-2.0036395147881314E7,
-2.0037471205137067E7,2.0036395147881314E7,2.0037471205137067E7,EPSG:3857
BOTTOM_LEFT
256,256
0,1,1,156412.0,156412.0
1,2,2,78206.0,78206.0
2,4,4,39103.0,39103.0
3,8,8,19551.5,19551.5
4,16,16,9775.75,9775.75
```

GDAL XML

Write a Pyramid to a GDAL XML File

```
Pyramid pyramid = Pyramid.createGlobalMercatorPyramid(maxZoom: 4)
GdalTmsPyramidWriter writer = new GdalTmsPyramidWriter()
String xml = writer.write(pyramid, serverUrl: 'https://myserver.com/${z}/${x}/${y}',
imageFormat: 'png')
println xml
```

```
<GDAL_WMS>
  <Service name='TMS'>
    <ServerURL>https://myserver.com/${z}/${x}/${y}</ServerURL>
    <SRS>EPSG:3857</SRS>
    <ImageFormat>png</ImageFormat>
  </Service>
  <DataWindow>
    <UpperLeftX>-2.0036395147881314E7</UpperLeftX>
    <UpperLeftY>2.003747120513706E7</UpperLeftY>
    <LowerRightX>2.0036395147881314E7</LowerRightX>
    <LowerRightY>-2.0037471205137067E7</LowerRightY>
    <TileLevel>4</TileLevel>
    <TileCountX>1</TileCountX>
    <TileCountY>1</TileCountY>
    <YOrigin>bottom</YOrigin>
  </DataWindow>
  <Projection>EPSG:3857</Projection>
  <BlockSizeX>256</BlockSizeX>
  <BlockSizeY>256</BlockSizeY>
  <BandsCount>3</BandsCount>
</GDAL_WMS>
```

Read a Pyramid from a GDAL XML File

```
        String xml = '''<GDAL_WMS>
  <Service name='TMS'>
    <ServerURL>https://myserver.com/${z}/${x}/${y}</ServerURL>
    <SRS>EPSG:3857</SRS>
    <ImageFormat>png</ImageFormat>
  </Service>
  <DataWindow>
    <UpperLeftX>-2.0036395147881314E7</UpperLeftX>
    <UpperLeftY>2.003747120513706E7</UpperLeftY>
    <LowerRightX>2.0036395147881314E7</LowerRightX>
    <LowerRightY>-2.0037471205137067E7</LowerRightY>
    <TileLevel>4</TileLevel>
    <TileCountX>1</TileCountX>
    <TileCountY>1</TileCountY>
    <YOrigin>bottom</YOrigin>
  </DataWindow>
  <Projection>EPSG:3857</Projection>
  <BlockSizeX>256</BlockSizeX>
  <BlockSizeY>256</BlockSizeY>
  <BandsCount>3</BandsCount>
</GDAL_WMS>'''
        GdalTmsPyramidReader reader = new GdalTmsPyramidReader()
        Pyramid pyramid = reader.read(xml)
```

```
geoscript.layer.Pyramid(proj:EPSG:3857, bounds:(-2.0036395147881314E7,-
2.0037471205137067E7,2.0036395147881314E7,2.003747120513706E7,EPSG:3857),
origin:BOTTOM_LEFT, tileWidth:256, tileHeight:256)
```

Generating Tiles

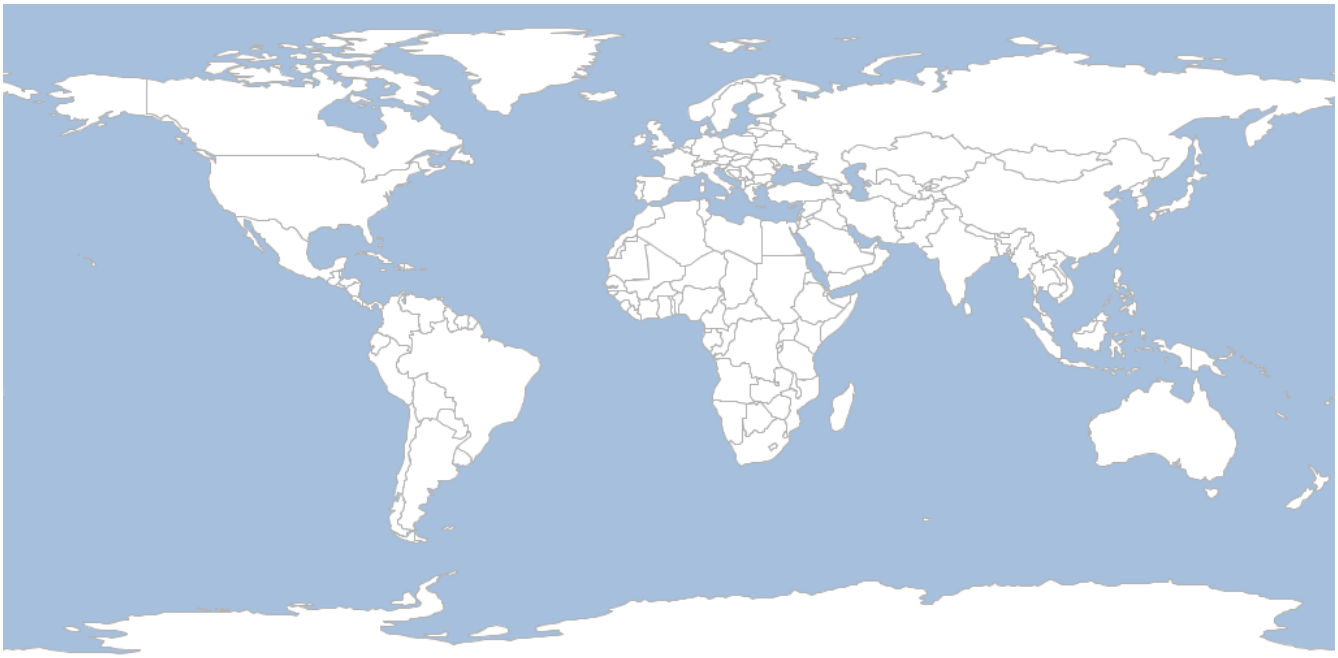
Generating Image Tiles

Generate Image Tiles to a MBTiles file

```
File file = new File("target/world.mbtiles")
MBTiles mbtiles = new MBTiles(file, "World", "World Tiles")

Workspace workspace = new GeoPackage('src/main/resources/data.gpkg')
Layer countries = workspace.get("countries")
countries.style = new Fill("#ffffff") + new Stroke("#b2b2b2", 0.5)
Layer ocean = workspace.get("ocean")
ocean.style = new Fill("#a5bfdd")

ImageTileRenderer renderer = new ImageTileRenderer(mbtiles, [ocean, countries])
TileGenerator generator = new TileGenerator()
generator.generate(mbtiles, renderer, 0, 2)
```

Generate Image Tiles to a GeoPackage file

```
File file = new File("target/world.gpkg")
geoscript.layer.GeoPackage geopackage = new geoscript.layer.GeoPackage(file, "World",
Pyramid.createGlobalGeodeticPyramid())

Workspace workspace = new GeoPackage('src/main/resources/data.gpkg')
Layer countries = workspace.get("countries")
countries.style = new Fill("#ffffff") + new Stroke("#b2b2b2", 0.5)
Layer ocean = workspace.get("ocean")
ocean.style = new Fill("#a5bfdd")

ImageTileRenderer renderer = new ImageTileRenderer(geopackage, [ocean, countries])
TileGenerator generator = new TileGenerator()
generator.generate(geopackage, renderer, 0, 2)
```



Tile Layer

Tile Layer Properties

Create a TileLayer from an MBTiles File.

```
File file = new File("src/main/resources/tiles.mbtiles")
MBTiles mbtiles = new MBTiles(file)
```

Get the TileLayer's name.

```
String name = mbtiles.name
println name
```

```
countries
```

Get the TileLayer's Bounds.

```
Bounds bounds = mbtiles.bounds
println bounds
```

```
(-2.0036395147881314E7,-
2.003747120513706E7,2.0036395147881314E7,2.003747120513706E7,EPSG:3857)
```

Get the TileLayer's Projection.

```
Projection proj = mbtiles.proj
println proj
```

```
EPSG:3857
```

Get the TileLayer's Pyramid.

```
Pyramid pyramid = mbtiles.pyramid
println pyramid
```

```
geoscript.layer.Pyramid(proj:EPSG:3857, bounds:(-2.0036395147881314E7,-
2.0037471205137067E7,2.0036395147881314E7,2.003747120513706E7,EPSG:3857),
origin:BOTTOM_LEFT, tileWidth:256, tileHeight:256)
```

Get a Tile from a TileLayer.

```
Tile tile = mbtiles.get(0, 0, 0)
println tile
```

```
Tile(x:0, y:0, z:0)
```

TileCursor

A TileCursor is a way to get a collection of Tiles from a TileLayer.

Get a TileCursor with all of the Tiles form a TileLayer in a zoom level.

```
File file = new File("src/main/resources/tiles.mbtiles")
MBTiles mbtiles = new MBTiles(file)

long zoomLevel = 1
TileCursor tileCursor = new TileCursor(mbtiles, zoomLevel)

println "Zoom Level: ${tileCursor.z}"
println "# of tiles: ${tileCursor.size}"
println "Bounds: ${tileCursor.bounds}"
println "Width / # Columns: ${tileCursor.width}"
println "Height / # Rows: ${tileCursor.height}"
println "MinX: ${tileCursor.minX}, MinY: ${tileCursor.minY}, MaxX: ${tileCursor.maxX},
MaxY: ${tileCursor.maxY}"

println "Tiles:"
tileCursor.each { Tile t ->
    println t
}
```

```
Zoom Level: 1
# of tiles: 4
Bounds: (-2.0036395147881314E7,-
2.0037471205137067E7,2.0036395147881314E7,2.003747120513706E7,EPSG:3857)
Width / # Columns: 2
Height / # Rows: 2
MinX: 0, MinY: 0, MaxX: 1, MaxY: 1

Tiles:
Tile(x:0, y:0, z:1)
Tile(x:1, y:0, z:1)
Tile(x:0, y:1, z:1)
Tile(x:1, y:1, z:1)
```

Get a TileCursor with Tiles form a TileLayer in a zoom level between min and max x and y coordinates.

```
File file = new File("src/main/resources/tiles.mbtiles")
MBTiles mbtiles = new MBTiles(file)

long zoomLevel = 4
long minX = 2
long minY = 4
long maxX = 5
long maxY = 8
TileCursor tileCursor = new TileCursor(mbtiles, zoomLevel, minX, minY, maxX, maxY)

println "Zoom Level: ${tileCursor.z}"
println "# of tiles: ${tileCursor.size}"
println "Bounds: ${tileCursor.bounds}"
println "Width / # Columns: ${tileCursor.width}"
println "Height / # Rows: ${tileCursor.height}"
println "MinX: ${tileCursor.minX}, MinY: ${tileCursor.minY}, MaxX: ${tileCursor.maxX},
MaxY: ${tileCursor.maxY}"

println "Tiles:"
tileCursor.each { Tile t ->
    println t
}
```

```
Zoom Level: 4
# of tiles: 20
Bounds: (-1.5027296360910986E7,-1.0018735602568535E7,-
5009098.786970329,2504683.900642129,EPSG:3857)
Width / # Columns: 4
Height / # Rows: 5
MinX: 2, MinY: 4, MaxX: 5, MaxY: 8

Tiles:
Tile(x:2, y:4, z:4)
Tile(x:3, y:4, z:4)
Tile(x:4, y:4, z:4)
Tile(x:5, y:4, z:4)
Tile(x:2, y:5, z:4)
Tile(x:3, y:5, z:4)
Tile(x:4, y:5, z:4)
Tile(x:5, y:5, z:4)
Tile(x:2, y:6, z:4)
Tile(x:3, y:6, z:4)
Tile(x:4, y:6, z:4)
Tile(x:5, y:6, z:4)
Tile(x:2, y:7, z:4)
Tile(x:3, y:7, z:4)
Tile(x:4, y:7, z:4)
Tile(x:5, y:7, z:4)
Tile(x:2, y:8, z:4)
Tile(x:3, y:8, z:4)
Tile(x:4, y:8, z:4)
Tile(x:5, y:8, z:4)
```

Get a TileCursor with Tiles form a TileLayer in a zoom level for a given Bounds.

```
File file = new File("src/main/resources/tiles.mbtiles")
MBTiles mbtiles = new MBTiles(file)

Bounds bounds = new Bounds(-102.875977, 45.433154, -96.481934, 48.118434,
"EPSG:4326").reproject("EPSG:3857")
int zoomLevel = 8
TileCursor tileCursor = new TileCursor(mbtiles, bounds, zoomLevel)

println "Zoom Level: ${tileCursor.z}"
println "# of tiles: ${tileCursor.size}"
println "Bounds: ${tileCursor.bounds}"
println "Width / # Columns: ${tileCursor.width}"
println "Height / # Rows: ${tileCursor.height}"
println "MinX: ${tileCursor.minX}, MinY: ${tileCursor.minY}, MaxX: ${tileCursor.maxX},
MaxY: ${tileCursor.maxY}"

println "Tiles:"
tileCursor.each { Tile t ->
    println t
}
```

```
Zoom Level: 8
# of tiles: 24
Bounds: (-1.1583540944868885E7,5635538.7764447965,-
1.0644334922311949E7,6261709.751605326,EPSG:3857)
Width / # Columns: 6
Height / # Rows: 4
MinX: 54, MinY: 164, MaxX: 59, MaxY: 167

Tiles:
Tile(x:54, y:164, z:8)
Tile(x:55, y:164, z:8)
Tile(x:56, y:164, z:8)
Tile(x:57, y:164, z:8)
Tile(x:58, y:164, z:8)
Tile(x:59, y:164, z:8)
Tile(x:54, y:165, z:8)
Tile(x:55, y:165, z:8)
Tile(x:56, y:165, z:8)
Tile(x:57, y:165, z:8)
Tile(x:58, y:165, z:8)
Tile(x:59, y:165, z:8)
Tile(x:54, y:166, z:8)
Tile(x:55, y:166, z:8)
Tile(x:56, y:166, z:8)
Tile(x:57, y:166, z:8)
Tile(x:58, y:166, z:8)
Tile(x:59, y:166, z:8)
Tile(x:54, y:167, z:8)
Tile(x:55, y:167, z:8)
Tile(x:56, y:167, z:8)
Tile(x:57, y:167, z:8)
Tile(x:58, y:167, z:8)
Tile(x:59, y:167, z:8)
```

Get a TileCursor with Tiles form a TileLayer in a zoom level for a given x and y resolution.

```
File file = new File("src/main/resources/tiles.mbtiles")
MBTiles mbtiles = new MBTiles(file)

Bounds bounds = new Bounds(-124.73142200000001, 24.955967, -66.969849, 49.371735,
"EPSG:4326").reproject("EPSG:3857")
double resolutionX = bounds.width / 400
double resolutionY = bounds.height / 300
TileCursor tileCursor = new TileCursor(mbtiles, bounds, resolutionX, resolutionY)

println "Zoom Level: ${tileCursor.z}"
println "# of tiles: ${tileCursor.size}"
println "Bounds: ${tileCursor.bounds}"
println "Width / # Columns: ${tileCursor.width}"
println "Height / # Rows: ${tileCursor.height}"
println "MinX: ${tileCursor.minX}, MinY: ${tileCursor.minY}, MaxX: ${tileCursor.maxX},
MaxY: ${tileCursor.maxY}"

println "Tiles:"
tileCursor.each { Tile t ->
    println t
}
```

```
Zoom Level: 4
# of tiles: 8
Bounds: (-1.5027296360910986E7,2504683.9006421305,-
5009098.786970329,7514051.701926393,EPSG:3857)
Width / # Columns: 4
Height / # Rows: 2
MinX: 2, MinY: 9, MaxX: 5, MaxY: 10

Tiles:
Tile(x:2, y:9, z:4)
Tile(x:3, y:9, z:4)
Tile(x:4, y:9, z:4)
Tile(x:5, y:9, z:4)
Tile(x:2, y:10, z:4)
Tile(x:3, y:10, z:4)
Tile(x:4, y:10, z:4)
Tile(x:5, y:10, z:4)
```