# Geoscript Groovy Cookbook

Jared Erickson

Version 1.13-SNAPSHOT

# Table of Contents


Introduction . . . 1
Using GeoScript . . . 1
- geoscript-groovy . . . 3
- Library . . . 4

Geometry Recipes . . . 5
- Creating Geometries . . . 5
- Points . . . 13
- LineStrings . . . 15
- Polygons . . . 27
- MultiLineStrings . . . 33
- MultiPolygons . . . 35
- Geometry Collections . . . 37
- Circular Strings . . . 40
- Circular Rings . . . 43
- Compound Curves . . . 48
- Compound Rings . . . 53
- Processing Geometries . . . 59
- Prepared Geometry . . . 112
- Reading and Writing Geometries . . . 124

Bounds Recipes . . . 138
- Creating Bounds . . . 138
- Getting Bounds Properties . . . 140
- Processing Bounds . . . 142

Projection Recipes . . . 153
- Creating Projections . . . 153
- Getting Projection Properties . . . 155
- Using Projections . . . 156
- Using Geodetic . . . 157
- Using Decimal Degrees . . . 158

Spatial Index Recipes . . . 162
- Using STRtree . . . 162
- Using Quadtree . . . 162
- Using GeoHash . . . 164

Viewer Recipes . . . 168
- Drawing geometries . . . 168
- Plotting geometries . . . 171

Plot Recipes . . . 174
- Processing Charts . . . 174

Creating Bar Charts . . . 178
Creating Pie Charts . . . 180
Creating Box Charts . . . 181
Creating Curve Charts . . . 181
Creating Regression Charts . . . 183
Creating Scatter Plot Charts . . . 184
Feature Recipes . . . 185
Creating Fields . . . 186
Creating Schemas . . . 187
Getting Schema Properties . . . 188
Getting Schema Fields . . . 189
Modifying Schemas . . . 190
Combining Schemas . . . 193
Creating Features from a Schema . . . 195
Reading and Writing Schemas . . . 196
Creating Features . . . 201
Getting Feature Properties . . . 203
Getting Feature Attributes . . . 204
Reading and Writing Features . . . 205
Filter Recipes . . . 217
Creating Filters . . . 217
Using Filters . . . 220
Evaluating Filters . . . 222
Creating Literals . . . 223
Creating Properties . . . 224
Evaluating Properties . . . 224
Creating Functions . . . 225
Evaluating Functions . . . 226
Process Functions . . . 227
Creating Colors . . . 229
Getting Color Formats . . . 231
Displaying Colors . . . 231
Using Color Palettes . . . 232
Creating Expressions from CQL . . . 237
Create Expression from static imports . . . 237
Process Recipes . . . 238
Execute a built-in Process . . . 238
Listing built-in Processes . . . 240
Executing a new Process . . . 242
Process Functions . . . 244
Render Recipes . . . 245

Creating Maps . . . 245
Map Properties . . . 250
Advanced Properties . . . 253
Rendering Maps . . . 256
Displaying Maps . . . 274
Drawing . . . 276
Plotting . . . 284
Style Recipes . . . 292
Creating Basic Styles . . . 292
Creating Strokes . . . 295
Creating Fills . . . 299
Creating Shapes . . . 302
Creating Icons . . . 304
Creating Labels . . . 305
Creating Transforms . . . 309
Creating Gradients . . . 311
Creating Unique Values . . . 313
Creating Color Maps . . . 314
Creating Channel Selection and Contrast Enhancement . . . 317
Reading and Writing Styles . . . 318
Workspace Recipes . . . 328
Using Workspaces . . . 328
Creating an in Memory Workspace . . . 330
Add Layer’s Features in Chunks . . . 331
Using a Directory Workspace . . . 332
Investigating Workspaces . . . 333
Creating Workspaces . . . 334
Database Workspace . . . 343
Layer Recipes . . . 347
Getting a Layer’s Properties . . . 347
Getting a Layer’s Features . . . 348
Adding, Updating, and Deleting . . . 355
Shapefiles . . . 362
Property . . . 365
Geoprocessing . . . 368
Layer Algebra . . . 375
Reading and Writing Layers . . . 382
Graticules . . . 406
Format Recipes . . . 411
Get a Format . . . 411
Get Names . . . 411

Read a Raster . . . 412
Write a Raster . . . 412
Check for a Raster . . . 413
GeoTIFF . . . 413
WorldImage . . . 414
Raster Recipes . . . 415
Raster Properties . . . 415
Raster Values . . . 419
Raster Processing . . . 421
Tile Recipes . . . 432
Tile . . . 432
Grid . . . 433
Pyramid . . . 433
Generating Tiles . . . 446
Tile Layer . . . 449
TileCursor . . . 462
OSM . . . 469

# 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.

# Using GeoScript

To use GeoScript Groovy you need Java, Java Advanced Imaging (JAI), and Groovy installed and on your PATH. Next, download the latest stable release, the latest in development build, or build the code yourself. Then put the GeoScript Groovy bin directory on your PATH. You are now ready to use GeoScript Groovy!

GeoScript Groovy has three commands:

1. geoscript-groovy (which can run Groovy files)

```
1. bash
Jared-Ericksons-MacBook-Pro:~ jericks$ cat script.groovy
import geoscript.geom.Point

p = new Point(-123.181458, 47.036439)
println p.wkt

poly = p.buffer(2.5)
println poly.wkt
Jared-Ericksons-MacBook-Pro:~ jericks$ geoscript-groovy script.groovy
POINT (-123.181458 47.036439)
POLYGON ((-120.681458 47.036439, -120.72949479899194 46.54871319495968, -120.871
7591687218 46.07973041908728, -121.10278396924365 45.647513417450995, -121.41369
104703364 45.26867204703363, -121.792532417451 44.95776496924364, -122.224749419
08729 44.72674016872178, -122.69373219495968 44.584475798991924, -123.181458 44.
536439, -123.66918380504033 44.584475798991924, -124.13816658091272 44.726740168
72178, -124.57038358254901 44.95776496924364, -124.94922495296638 45.26867204703
363, -125.26013203075637 45.647513417450995, -125.49115683127822 46.079730419087
28, -125.63342120100808 46.54871319495968, -125.681458 47.036439, -125.633421201
00808 47.52416480504032, -125.49115683127822 47.993147580912726, -125.2601320307
5637 48.42536458254901, -124.94922495296638 48.80420595296637, -124.570383582549
01 49.11511303075637, -124.13816658091272 49.34613783127822, -123.66918380504032
 49.48840220100808, -123.181458 49.536439, -122.69373219495968 49.48840220100808
, -122.22474941908727 49.34613783127821, -121.792532417451 49.11511303075636, -1
21.41369104703364 48.80420595296636, -121.10278396924363 48.425364582549, -120.8
7175916872178 47.99314758091272, -120.72949479899192 47.524164805040314, -120.68
1458 47.036439))
Jared-Ericksons-MacBook-Pro:~ jericks$
```

1. geoscript-groovysh (which starts a REPL shell)

```
1. java
Jared-Ericksons-MacBook-Pro:~ jericks$ geoscript-groovysh
Groovy Shell (2.4.10, JVM: 1.8.0_31)
Type ':help' or ':h' for help.
--------------------------------------------------------------------------------
groovy:000> import geoscript.geom.Point
===> geoscript.geom.Point
groovy:000> p = new Point(-123.181458,47.036439)
===> POINT (-123.181458 47.036439)
groovy:000> p.buffer(2.5)
===> POLYGON ((-120.681458 47.036439, -120.72949479899194 46.54871319495968, -12
0.8717591687218 46.07973041908728, -121.10278396924365 45.647513417450995, -121.
41369104703364 45.26867204703363, -121.792532417451 44.95776496924364, -122.2247
4941908729 44.72674016872178, -122.69373219495968 44.584475798991924, -123.18145
8 44.536439, -123.66918380504033 44.584475798991924, -124.13816658091272 44.7267
4016872178, -124.57038358254901 44.95776496924364, -124.94922495296638 45.268672
04703363, -125.26013203075637 45.647513417450995, -125.49115683127822 46.0797304
1908728, -125.63342120100808 46.54871319495968, -125.681458 47.036439, -125.6334
2120100808 47.52416480504032, -125.49115683127822 47.993147580912726, -125.26013
203075637 48.42536458254901, -124.94922495296638 48.80420595296637, -124.5703835
8254901 49.11511303075637, -124.13816658091272 49.34613783127822, -123.669183805
04032 49.48840220100808, -123.181458 49.536439, -122.69373219495968 49.488402201
00808, -122.22474941908727 49.34613783127821, -121.792532417451 49.1151130307563
6, -121.41369104703364 48.80420595296636, -121.10278396924363 48.425364582549, -
120.87175916872178 47.99314758091272, -120.72949479899192 47.524164805040314, -1
20.681458 47.036439))
groovy:000>
```

1. geoscript-groovyConsole (which starts a graphical editor/mini IDE)

# geoscript-groovy

The geoscript-groovy command can run scripts file but it can also run inline scripts.

*Convert a shapefile to geojson*

```
geoscript-groovy -e "println new
geoscript.layer.Shapefile('states.shp').toJSONString()"
```

*Get the Bounds of a Shapefile as a Geometry*

```
geoscript-groovy -e "println new
geoscript.layer.Shapefile('states.shp').bounds.geometry"
```

*Count the number of Features in a Shapefile*

```
geoscript-groovy -e "println new geoscript.layer.Shapefile('states.shp').count"
```

*Render a Shapefile to an image*

```
geoscript-groovy -e "geoscript.render.Draw.draw(new
geoscript.layer.Shapefile('states.shp'), out: 'image.png')"
```

*Pipe a Shapefile's geometry to another command line application that buffers each feature*

```
geoscript-groovy -e "new geoscript.layer.Shapefile('states.shp').eachFeature{ println
it.geom.centroid}" | geom combine | geom buffer -d 1.5
```

*Pipe the results of buffering a point to convert it to KML*

```
echo "POINT (1 1)" | geom buffer -d 10 | geoscript-groovy -e "println
geoscript.geom.Geometry.fromWKT(System.in.text).kml"
```

# Library

## Maven

*You can also use GeoScript Groovy as a library. If you use Maven you will need to add the Boundless Maven Repository:*

```
<repositories>
    <repository>
        <id>boundless</id>
        <name>Boundless Maven Repository</name>
        <url>https://repo.boundlessgeo.com/main</url>
        <snapshots>
            <enabled>true</enabled>
        </snapshots>
    </repository>
</repositories>
```

*and then include the GeoScript Groovy dependency:*

```
<dependency>
    <groupId>org.geoscript</groupId>
    <artifactId>geoscript-groovy</artifactId>
    <version>1.13-SNAPSHOT</version>
</dependency>
```

## Gradle

*To use GeoScript Groovy in a Gradle project, add the following repositories:*

```
repositories {
    maven {
        url "https://repo.boundlessgeo.com/main"
    }
    maven {
        url "http://download.osgeo.org/webdav/geotools/"
    }
}
```

*and the include the GeoScript Groovy dependency:*

```
dependencies {
    compile("org.geoscript:geoscript-groovy:1.13-SNAPSHOT")
}
```

# Geometry Recipes

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

## Creating Geometries

### Point

*Create a Point with an XY*

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

### LineString

*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]
)
```

## Polygon

*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 Polygon with holes*

```
Polygon polygonWithHoles = new Polygon(
    // Exterior Ring
    new LinearRing(
        [-122.39138603210449, 47.58659965790016],
        [-122.41250038146973, 47.57681522195182],
        [-122.40305900573729, 47.56523364515569],
        [-122.38117218017578, 47.56621817878201],
        [-122.3712158203125, 47.57235661809739],
        [-122.37602233886717, 47.584747123985615],
        [-122.39138603210449, 47.58659965790016]
    ),
    // Holes
    [
        new LinearRing(
            [-122.39859580993652, 47.578957532923376],
            [-122.40468978881836, 47.57548347095205],
            [-122.39593505859376, 47.570271945800094],
            [-122.3920726776123, 47.57606249728773],
            [-122.39859580993652, 47.578957532923376]
        ),
        new LinearRing(
            [-122.3836612701416, 47.58156292813543],
            [-122.38829612731934, 47.57114056934196],
            [-122.37456321716309, 47.57420959047542],
            [-122.37868309020995, 47.58023129789275],
            [-122.3836612701416, 47.58156292813543]
        )
    ]
)
```

## MultiPoint

*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)
])
```

## MultiLineString

*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]
        )
])
```

## MultiPolygon

*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]
    ]])
)
```

## GeometryCollection

*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)
)
```

## CircularString

*Create a CircularString with a List of Points*

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

## CircularRing

*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]
])
```

## CompoundCurve

*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]

    ])
])
```

## CompoundRing

*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],
        ])
])
```

# Points

*Get x, y, and z values from a Point*

```
Point point = new Point(-122.38632, 47.58208, 101.45)
println "X = ${point.x}"
println "Y = ${point.y}"
println "Z = ${point.z}"
```

```
X = -122.38632
Y = 47.58208
Z = 101.45
```

*Add two Points together to create a MultiPoint*

```
Point point1 = new Point(-122.38632, 47.58208)
Point point2 = new Point(-122.37001, 47.55868)
MultiPoint points = point1 + point2
```

*Add a Point to a MultiPoint*

```
MultiPoint multiPoint = new MultiPoint(
        new Point(-122.83813,47.05141),
        new Point(-122.38220,47.58023)
)
println multiPoint.wkt
MultiPoint newMultiPoint = multiPoint + new Point(-122.48657, 47.271775)
println newMultiPoint.wkt
```

```
MULTIPOINT ((-122.83813 47.05141), (-122.3822 47.58023))
MULTIPOINT ((-122.83813 47.05141), (-122.3822 47.58023), (-122.48657 47.271775))
```

MultiPoint

MultiPoint with extra Point

*Calculate the angle between two points*

```
Point point1 = new Point(-122.29980, 47.65058)
Point point2 = new Point(-120.54199, 46.64943)
double angleInDegrees = point1.getAngle(point2, "degrees")
println "Angle in degrees = ${angleInDegrees}"

double angleInRadians = point1.getAngle(point2, "radians")
println "Angle in radians = ${angleInRadians}"
```

```
Angle in degrees = -29.663413013476646
Angle in radians = -0.5177242244641005
```

*Calculate the azimuth between two points*

```
Point point1 = new Point(-122.29980, 47.65058)
Point point2 = new Point(-120.54199, 46.64943)
double azimuth = point1.getAzimuth(point2)
println "Azimuth = ${azimuth}"
```

```
Azimuth = 129.21026122904846
```

# LineStrings

*Get the start Point from a LineString*

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

*Get the end Point from a LineString*

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

*Reverse a LineString*

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

LineString reversedLineString = lineString.reverse()
Point reversedStartPoint = reversedLineString.startPoint
```

Original LineString showing start point

Reversed LineString showing start point

*Determine if a LineString is closed or not*

```
LineString lineString1 = new LineString(
        [3.1982421875, 43.1640625],
        [6.7138671875, 49.755859375],
        [9.7021484375, 42.5927734375],
        [15.3271484375, 53.798828125]
)
boolean isClosed1 = lineString1.closed
println "Is ${lineString1.wkt} closed? ${isClosed1}"

LineString lineString2 = new LineString(
        [3.1982421875, 43.1640625],
        [6.7138671875, 49.755859375],
        [9.7021484375, 42.5927734375],
        [15.3271484375, 53.798828125],
        [3.1982421875, 43.1640625]
)
boolean isClosed2 = lineString2.closed
println "Is ${lineString2.wkt} closed? ${isClosed2}"
```

```
Is LINESTRING (3.1982421875 43.1640625, 6.7138671875 49.755859375, 9.7021484375
42.5927734375, 15.3271484375 53.798828125) closed? false
```

```
Is LINESTRING (3.1982421875 43.1640625, 6.7138671875 49.755859375, 9.7021484375
42.5927734375, 15.3271484375 53.798828125, 3.1982421875 43.1640625) closed? true
```

*Determine if a LineString is a Ring or not*

```
LineString lineString1 = new LineString(
    [-122.391428, 47.563300],
    [-122.391836, 47.562793],
    [-122.391010, 47.562417],
    [-122.390516, 47.563126]
)
boolean isRing1 = lineString1.ring
println "Is ${lineString1.wkt} a ring? ${isRing1}"

LineString lineString2 = new LineString(
    [-122.391428, 47.563300],
    [-122.391836, 47.562793],
    [-122.391010, 47.562417],
    [-122.390516, 47.563126],
    [-122.391428, 47.563300]
)
boolean isRing2 = lineString2.ring
println "Is ${lineString2.wkt} a ring? ${isRing2}"
```

```
Is LINESTRING (-122.391428 47.5633, -122.391836 47.562793, -122.39101 47.562417,
-122.390516 47.563126) a ring? false
```

```
Is LINESTRING (-122.391428 47.5633, -122.391836 47.562793, -122.39101 47.562417,
-122.390516 47.563126, -122.391428 47.5633) a ring? true
```

*Close an open LineString to create a LinearRing*

```
LineString lineString = new LineString(
        [-122.391428, 47.563300],
        [-122.391836, 47.562793],
        [-122.391010, 47.562417],
        [-122.390516, 47.563126]
)

LinearRing linearRing = lineString.close()
```

Open LineString

Closed LinearRing

*Add a LineString to another LineString to create a MultiLineString*

```
LineString lineString1 = new LineString([
    [-122.39142894744873, 47.5812734461813],
    [-122.38237380981445, 47.58121554959838]
])

LineString lineString2 = new LineString([
    [-122.38640785217285, 47.58552866972616],
    [-122.38670825958253, 47.57837853860192]
])

MultiLineString multiLineString = lineString1 + lineString2
```

*Add a Point to a  LineString*

```
LineString lineString = new LineString([
        [-122.39142894744873, 47.5812734461813],
        [-122.38237380981445, 47.58121554959838]
])

Point point = new Point(-122.38640785217285, 47.58552866972616)

LineString lineStringWithPoint = lineString + point
```

*Add a Point to a LineString at a specific index*

```
LineString lineString = new LineString([
        [-122.39142894744873, 47.5812734461813],
        [-122.38237380981445, 47.58121554959838]
])

Point point = new Point(-122.38640785217285, 47.58552866972616)

LineString lineStringWithPoint = lineString.addPoint(1, point)
```

*Set a Point on a LineString at a specific index*

```
LineString lineString = new LineString([
        [-122.39142894744873, 47.5812734461813],
        [-122.38237380981445, 47.58121554959838]
])

Point point = new Point(-122.38640785217285, 47.58552866972616)

LineString newLineString = lineString.setPoint(1, point)
```

*Remove a Point on a LineString at a specific index*

```
LineString lineString = new LineString([
        [-122.39142894744873, 47.5812734461813],
        [-122.38237380981445, 47.58121554959838],
        [-122.38640785217285, 47.58552866972616]
])

LineString newLineString = lineString.removePoint(2)
```

*Remove a Point from the end of a LineString.*

```
LineString lineString1 = new LineString([
    [-122.39423990249632, 47.57926150237904],
    [-122.3918581008911, 47.58121554959838],
    [-122.38657951354979, 47.58121554959838],
    [-122.38638639450075, 47.58535499390333],
    [-122.38374710083008, 47.58535499390333]
])
```

```
LineString lineString2 = -lineString1
```

```
LineString lineString3 = -lineString2
```

*Interpolate Points on a LineString*

```
LineString lineString = new LineString([
    [-122.39423990249632, 47.57926150237904],
    [-122.3918581008911, 47.58121554959838],
    [-122.38657951354979, 47.58121554959838],
    [-122.38638639450075, 47.58535499390333],
    [-122.38374710083008, 47.58535499390333]
])
Point startPoint = lineString.interpolatePoint(0.0)
Point midPoint = lineString.interpolatePoint(0.5)
Point endPoint = lineString.interpolatePoint(1.0)
```

*Locate the position of Points on a LineString*

```
LineString lineString = new LineString([
    [-122.39423990249632, 47.57926150237904],
    [-122.3918581008911, 47.58121554959838],
    [-122.38657951354979, 47.58121554959838],
    [-122.38638639450075, 47.58535499390333],
    [-122.38374710083008, 47.58535499390333]
])

Point point1 = new Point(-122.39423990249632, 47.57926150237904)
Double position1 = lineString.locatePoint(point1)
println "Position of ${point1} is ${position1}"

Point point2 = new Point(-122.38736758304911, 47.58121554959838)
Double position2 = lineString.locatePoint(point2)
println "Position of ${point2} is ${position2}"

Point point3 = new Point(-122.38374710083008, 47.58535499390333)
Double position3 = lineString.locatePoint(point3)
println "Position of ${point3} is ${position3}"
```

```
Position of POINT (-122.39423990249632 47.57926150237904) is 0.0
Position of POINT (-122.38736758304911 47.58121554959838) is 0.5000000000004425
Position of POINT (-122.38374710083008 47.58535499390333) is 1.0
```

*Place Points on a LineString*

```
LineString lineString = new LineString ([
    [-122.37155914306639, 47.57166173655188],
    [-122.32160568237306, 47.5714301073211]
])

Point point1 = new Point(-122.358341217041, 47.57432539907205)
Point pointOnLine1 = lineString.placePoint(point1)

Point point2 = new Point(-122.33860015869139, 47.56830301243495)
Point pointOnLine2 = lineString.placePoint(point2)
```

Points near LineString

Point on LineString

*Extract part of a LineString*

```
LineString lineString = new LineString([
    [-122.39423990249632, 47.57926150237904],
    [-122.3918581008911, 47.58121554959838],
    [-122.38657951354979, 47.58121554959838],
    [-122.38638639450075, 47.58535499390333],
    [-122.38374710083008, 47.58535499390333]
])
LineString subLine = lineString.subLine(0.33, 0.66)
```

# Polygons

*Get a Polygon's exterior LinearRing*

```
Polygon polygon = new Polygon(
    // Exterior Ring
    new LinearRing(
        [-122.39138603210449, 47.58659965790016],
        [-122.41250038146973, 47.57681522195182],
        [-122.40305900573729, 47.56523364515569],
        [-122.38117218017578, 47.56621817878201],
        [-122.3712158203125, 47.57235661809739],
        [-122.37602233886717, 47.584747123985615],
        [-122.39138603210449, 47.58659965790016]
    ),
    // Holes
    [
        new LinearRing(
            [-122.39859580993652, 47.578957532923376],
            [-122.40468978881836, 47.57548347095205],
            [-122.39593505859376, 47.570271945800094],
            [-122.3920726776123, 47.57606249728773],
            [-122.39859580993652, 47.578957532923376]
        ),
        new LinearRing(
            [-122.3836612701416, 47.58156292813543],
            [-122.38829612731934, 47.57114056934196],
            [-122.37456321716309, 47.57420959047542],
            [-122.37868309020995, 47.58023129789275],
            [-122.3836612701416, 47.58156292813543]
        )
    ]
)
LinearRing exteriorRing = polygon.getExteriorRing()
```

*Get a Polygon's interior LinearRings*

```
Polygon polygon = new Polygon(
        // Exterior Ring
        new LinearRing(
                [-122.39138603210449, 47.58659965790016],
                [-122.41250038146973, 47.57681522195182],
                [-122.40305900573729, 47.56523364515569],
                [-122.38117218017578, 47.56621817878201],
                [-122.3712158203125, 47.57235661809739],
                [-122.37602233886717, 47.584747123985615],
                [-122.39138603210449, 47.58659965790016]
        ),
        // Holes
        [
                new LinearRing(
                        [-122.39859580993652, 47.578957532923376],
                        [-122.40468978881836, 47.57548347095205],
                        [-122.39593505859376, 47.570271945800094],
                        [-122.3920726776123, 47.57606249728773],
                        [-122.39859580993652, 47.578957532923376]
                ),
                new LinearRing(
                        [-122.3836612701416, 47.58156292813543],
                        [-122.38829612731934, 47.57114056934196],
                        [-122.37456321716309, 47.57420959047542],
                        [-122.37868309020995, 47.58023129789275],
                        [-122.3836612701416, 47.58156292813543]
                )
        ]
)

println "# Interior Rings = ${polygon.numInteriorRing}"
(0..<polygon.numInteriorRing).each { int i ->
    println "  ${polygon.getInteriorRingN(i)}"
}

println "Interior Rings"
polygon.interiorRings.each { LinearRing ring ->
    println "  ${ring}"
}
```

```
# Interior Rings = 2
  LINEARRING (-122.39859580993652 47.578957532923376, -122.40468978881836
47.57548347095205, -122.39593505859376 47.570271945800094, -122.3920726776123
47.57606249728773, -122.39859580993652 47.578957532923376)
  LINEARRING (-122.3836612701416 47.58156292813543, -122.38829612731934
47.57114056934196, -122.37456321716309 47.57420959047542, -122.37868309020995
47.58023129789275, -122.3836612701416 47.58156292813543)

Interior Rings
  LINEARRING (-122.39859580993652 47.578957532923376, -122.40468978881836
47.57548347095205, -122.39593505859376 47.570271945800094, -122.3920726776123
47.57606249728773, -122.39859580993652 47.578957532923376)
  LINEARRING (-122.3836612701416 47.58156292813543, -122.38829612731934
47.57114056934196, -122.37456321716309 47.57420959047542, -122.37868309020995
47.58023129789275, -122.3836612701416 47.58156292813543)
```

*Add a Polygon to another Polygon to create a MultiPolygon*

```
Polygon polygon1 = new Polygon ([[
  [-122.2723388671875,  47.818687628247105],
  [-122.37945556640624, 47.66168780332917],
  [-121.95373535156249, 47.67093619422418],
  [-122.2723388671875,  47.818687628247105]
]])
Polygon polygon2 = new Polygon ([[
  [-122.76672363281249, 47.42437092240516],
  [-122.76672363281249, 47.59505101193038],
  [-122.52227783203125, 47.59505101193038],
  [-122.52227783203125, 47.42437092240516],
  [-122.76672363281249, 47.42437092240516]
]])
MultiPolygon multiPolygon = polygon1 + polygon2
```

*Split a Polygon with a LineString*

```
Polygon polygon = new Polygon(
    new LinearRing(
        [-122.39138603210449, 47.58659965790016],
        [-122.41250038146973, 47.57681522195182],
        [-122.40305900573729, 47.56523364515569],
        [-122.38117218017578, 47.56621817878201],
        [-122.3712158203125, 47.57235661809739],
        [-122.37602233886717, 47.584747123985615],
        [-122.39138603210449, 47.58659965790016]
    )
)

LineString lineString = new LineString([
    [-122.3924160003662, 47.56395951534652],
    [-122.38649368286131, 47.58729434121508]
])

MultiPolygon multiPolygon = polygon.split(lineString)
```

## MultiLineStrings

*Add a LineString to a MultiLineString to get a new MultiLineString*

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

LineString lineString = new LineString (
    [-122.29705810546874, 47.303447043862626],
    [-122.42889404296875, 47.23262467463881]
)

MultiLineString newMultiLineString = multiLineString + lineString
```

*Merge the LineStrings in a MultiLineString*

```
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]
    )
])
MultiLineString mergedMultiLineString = multiLineString.merge()

println "Original MultiLineString = ${multiLineString}"
println "Merged MultiLineString   = ${mergedMultiLineString}"
```

```
Original MultiLineString = MULTILINESTRING ((-122.3822021484375 47.57837853860192,
-122.32452392578125 47.48380086737799), (-122.32452392578125 47.48380086737799,
-122.29705810546874 47.303447043862626), (-122.29705810546874 47.303447043862626,
-122.42889404296875 47.23262467463881))

Merged MultiLineString   = MULTILINESTRING ((-122.3822021484375 47.57837853860192,
-122.32452392578125 47.48380086737799, -122.29705810546874 47.303447043862626,
-122.42889404296875 47.23262467463881))
```

# MultiPolygons

*Add a Polygon to a MultiPolygon to get a new MultiPolygon*

```
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]
    ]])
)
Polygon polygon = new Polygon ([[
    [-122.32177734375, 47.54501765940571],
    [-122.27645874023438, 47.36673410912714],
    [-122.03887939453125, 47.44480754169437],
    [-122.15972900390624, 47.55150616084034],
    [-122.32177734375, 47.54501765940571]
]])

MultiPolygon newMultiPolygon = multiPolygon + polygon
```

*Split a MultiPolygon with a LineString*

```
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 ([[
          [-122.32177734375, 47.54501765940571],
          [-122.27645874023438, 47.36673410912714],
          [-122.03887939453125, 47.44480754169437],
          [-122.15972900390624, 47.55150616084034],
          [-122.32177734375, 47.54501765940571]
    ]])
)

LineString lineString = new LineString([
    [-122.84362792968749, 47.484728927366504],
    [-122.05810546875, 47.50421439972969],
    [-122.35748291015625, 47.85832433461554]
])

Geometry splitGeometry = multiPolygon.split(lineString)
```

Before

After

# Geometry Collections

*Add a Geometry to a Geometry Collection*

```
GeometryCollection geometryCollection = new GeometryCollection([
    new Point(-122.38654196262358, 47.581211931059826),
    new LineString([
        [-122.3865446448326, 47.58118841055313],
        [-122.38657146692276, 47.58067638459562]
    ]),
    new Polygon(new LinearRing([
        [-122.38693356513977, 47.58088445228483],
        [-122.38672703504562, 47.58088445228483],
        [-122.38672703504562, 47.58096225129535],
        [-122.38693356513977, 47.58096225129535],
        [-122.38693356513977, 47.58088445228483]
    ]))
])
GeometryCollection newGeometryCollection =  geometryCollection + new Point(-
122.38718032836913, 47.58121374032914)
```

Original Geometry Collection

New Geometry Collection

*Get a subset of Geometries from a Geometry Collection*

```
GeometryCollection geometryCollection = new GeometryCollection([
        new Point(-122.38654196262358, 47.581211931059826),
        new LineString([
                [-122.3865446448326, 47.58118841055313],
                [-122.38657146692276, 47.58067638459562]
        ]),
        new Polygon(new LinearRing([
                [-122.38693356513977, 47.58088445228483],
                [-122.38672703504562, 47.58088445228483],
                [-122.38672703504562, 47.58096225129535],
                [-122.38693356513977, 47.58096225129535],
                [-122.38693356513977, 47.58088445228483]
        ]))
])
```

```
Polygon slicedGeometryCollection1 =  geometryCollection.slice(2)
```

```
GeometryCollection slicedGeometryCollection2 =  geometryCollection.slice(0,2)
```

*Get a Geometry from a GeometryCollection that is the most type specific as possible.*

```
GeometryCollection geometryCollection1 = new GeometryCollection([
    new Point(-122.38654196262358, 47.581211931059826),
    new Point(-122.38718032836913, 47.58121374032914),
])
Geometry narrowedGeometry1 = geometryCollection1.narrow()
println "Narrow Geometry #1 = ${narrowedGeometry1.wkt}"
```

```
Narrow Geometry #1 = MULTIPOINT ((-122.38654196262358 47.581211931059826), (-
122.38718032836913 47.58121374032914))
```

```
GeometryCollection geometryCollection2 = new GeometryCollection([
        new Point(-122.38654196262358, 47.581211931059826),
        new Polygon(new LinearRing([
                [-122.38693356513977, 47.58088445228483],
                [-122.38672703504562, 47.58088445228483],
                [-122.38672703504562, 47.58096225129535],
                [-122.38693356513977, 47.58096225129535],
                [-122.38693356513977, 47.58088445228483]
        ]))
])
Geometry narrowedGeometry2 = geometryCollection2.narrow()
println "Narrow Geometry #2 = ${narrowedGeometry2.wkt}"
```

```
Narrow Geometry #2 = GEOMETRYCOLLECTION (POINT (-122.38654196262358
47.581211931059826), POLYGON ((-122.38693356513977 47.58088445228483,
-122.38672703504562 47.58088445228483, -122.38672703504562 47.58096225129535,
-122.38693356513977 47.58096225129535, -122.38693356513977 47.58088445228483)))
```

## Circular Strings

*Get Curved WKT from a Circular Strings*

```
CircularString circularString = new CircularString([
        [-122.464599609375, 47.247542522268006],
        [-122.03613281249999, 47.37789454155521],
        [-122.37670898437499, 47.58393661978134]
])
println "WKT = ${circularString.wkt}"
println "Curnved WKT = ${circularString.curvedWkt}"
```

```
WKT = LINESTRING (-122.464599609375 47.247542522268006, -122.4550237920096
47.23410579860605, -122.4348780833765 47.21113402059009, -122.41190630536055
47.19098831195699, -122.38650151145254 47.17401337136692, -122.3590983847198
47.16049964475972, -122.33016580025833 47.15067835574435, -122.30019880260986
47.144717549298825, -122.26971013541258 47.1427192164741, -122.2392214682153
47.144717549298825, -122.20925447056683 47.15067835574435, -122.18032188610536
47.16049964475972, -122.15291875937262 47.17401337136692, -122.12751396546462
47.19098831195699, -122.10454218744866 47.21113402059009, -122.08439647881556
47.23410579860605, -122.06742153822549 47.259510592514054, -122.05390781161829
47.28691371924678, -122.04408652260291 47.31584630370827, -122.0381257161574
47.34581330135674, -122.03612738333267 47.37630196855401, -122.03613281249999
47.37789454155521, -122.0381257161574 47.40679063575128, -122.04408652260291
47.43675763339975, -122.05390781161829 47.46569021786124, -122.06742153822549
47.493093344593966, -122.08439647881556 47.51849813850197, -122.10454218744866
47.54146991651793, -122.12751396546462 47.56161562515103, -122.15291875937262
47.5785905657411, -122.18032188610536 47.5921042923483, -122.20925447056683
47.60192558136367, -122.2392214682153 47.607886387809195, -122.26971013541258
47.60988472063392, -122.30019880260986 47.607886387809195, -122.33016580025833
47.60192558136367, -122.3590983847198 47.5921042923483, -122.37670898437499
47.58393661978134)
Curnved WKT = CIRCULARSTRING (-122.464599609375 47.247542522268006,
-122.03613281249999 47.37789454155521, -122.37670898437499 47.58393661978134)
```

*Get control points from a Circular Strings*

```
CircularString circularString = new CircularString([
        [-122.464599609375, 47.247542522268006],
        [-122.03613281249999, 47.37789454155521],
        [-122.37670898437499, 47.58393661978134]
])
List<Point> points = circularString.controlPoints
points.each { Point point ->
    println point
}
```

```
POINT (-122.464599609375 47.247542522268006)
POINT (-122.03613281249999 47.37789454155521)
POINT (-122.37670898437499 47.58393661978134)
```

*Convert a Circural Strings to a Linear Geometry*

```
CircularString circularString = new CircularString([
        [-122.464599609375, 47.247542522268006],
        [-122.03613281249999, 47.37789454155521],
        [-122.37670898437499, 47.58393661978134]
])
Geometry linear = circularString.linear
println linear.wkt
```

```
LINESTRING (-122.464599609375 47.247542522268006, -122.4550237920096
47.23410579860605, -122.4348780833765 47.21113402059009, -122.41190630536055
47.19098831195699, -122.38650151145254 47.17401337136692, -122.3590983847198
47.16049964475972, -122.33016580025833 47.15067835574435, -122.30019880260986
47.144717549298825, -122.26971013541258 47.1427192164741, -122.2392214682153
47.144717549298825, -122.20925447056683 47.15067835574435, -122.18032188610536
47.16049964475972, -122.15291875937262 47.17401337136692, -122.12751396546462
47.19098831195699, -122.10454218744866 47.21113402059009, -122.08439647881556
47.23410579860605, -122.06742153822549 47.259510592514054, -122.05390781161829
47.28691371924678, -122.04408652260291 47.31584630370827, -122.0381257161574
47.34581330135674, -122.03612738333267 47.37630196855401, -122.03613281249999
47.37789454155521, -122.0381257161574 47.40679063575128, -122.04408652260291
47.43675763339975, -122.05390781161829 47.46569021786124, -122.06742153822549
47.493093344593966, -122.08439647881556 47.51849813850197, -122.10454218744866
47.54146991651793, -122.12751396546462 47.56161562515103, -122.15291875937262
47.5785905657411, -122.18032188610536 47.5921042923483, -122.20925447056683
47.60192558136367, -122.2392214682153 47.607886387809195, -122.26971013541258
47.60988472063392, -122.30019880260986 47.607886387809195, -122.33016580025833
47.60192558136367, -122.3590983847198 47.5921042923483, -122.37670898437499
47.58393661978134)
```

# Circular Rings

*Get Curved WKT from a Circular Ring*

```
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]
])
println "WKT = ${circularRing.wkt}"
println "Curnved WKT = ${circularRing.curvedWkt}"
```

```
WKT = LINEARRING (-118.47656249999999 41.508577297439324, -118.58869123015452
41.901774579059115, -118.91479847512254 43.54122641035517, -119.02412442253558
45.209218040904574, -118.91479847512252 46.877209671453976, -118.58869123015451
48.516661502750026, -118.05138247300694 50.09952205941734, -117.31206570548474
51.59870815847947, -116.38339084245754 52.98856831012856, -115.28124776830906
54.245321621827856, -114.02449445660976 55.347464695976335, -112.63463430496067
56.27613955900354, -111.13544820589854 57.01545632652574, -109.6875 57.51582286553883,
-109.55258764923123 57.552765083673314, -107.91313581793517 57.87887232864133,
-106.24514418738576 57.98819827605437, -104.57715255683635 57.87887232864133,
-102.9377007255403 57.552765083673314, -101.35484016887298 57.01545632652574,
-99.85565406981085 56.27613955900354, -98.46579391816176 55.347464695976335,
-97.20904060646247 54.245321621827856, -96.10689753231398 52.98856831012856,
-95.17822266928678 51.59870815847947, -94.43890590176458 50.09952205941733,
-93.901597144617 48.51666150275002, -93.575489899649 46.87720967145396,
-93.46616395223595 45.20921804090456, -93.575489899649 43.54122641035516, -93.8671875
42.032974332441405, -92.52944589377746 42.87620255408929, -90.13998424943787
44.0545546346726, -87.61715897053445 44.91093841996455, -85.00413629704853
45.43070094596436, -82.34562577139025 45.60494893174677, -79.68711524573196
45.43070094596437, -77.07409257224604 44.910938419964566, -74.55126729334262
44.05455463467261, -72.16180564900301 42.876202554089296, -69.94659199044581
41.396044109014326, -67.9435292375422 39.639405220820365, -66.18689034934823
37.63634246791675, -64.70673190427325 35.42112880935955, -63.528379823689946
33.03166716501995, -62.67199603839799 30.50884188611652, -62.57812500000001
30.14512718337613, -62.15223351239818 27.895819212630613, -61.97798552661578
25.237308686972327, -62.15223351239818 22.578798161314037, -62.671996038397985
19.965775487828132, -63.52837982368993 17.442950208924703, -64.70673190427325
15.0534885645851, -66.18689034934822 12.838274906027902, -67.94352923754218
10.835212153124278, -69.9465919904458 9.078573264930323, -72.16180564900299
7.598414819855346, -74.55126729334259 6.42006273927203, -77.07409257224602
5.563678953980077, -79.68711524573193 5.0439164279802675, -82.34562577139022
4.869668442197863, -85.00413629704852 5.043916427980264, -87.61715897053442
5.5636789539800695, -90.13998424943784 6.420062739272023, -92.10937499999999
7.36246686553575, -94.00832592648507 5.722586433241059, -96.3693387982898
4.1450080684486466, -98.91606817456314 2.88910012519991, -101.60493880959022
1.9763515366073037, -104.38994338130999 1.4223796840833636, -107.22342968935115
1.2366631796147907, -110.05691599739232 1.4223796840833813, -112.84192056911208
1.9763515366073356, -115.53079120413916 2.8891001251999597, -118.07752058041248
4.1450080684487105, -120.43853345221721 5.722586433241133, -122.57343223472037
7.594842416368293, -124.4456882178475 9.729741198871439, -126.02326658263992
12.09075407067618, -127.265625 14.604847155053898, -127.27917452588866
14.637483446949503, -128.19192311448128 17.326354081976593, -128.7458949670052
20.111358653696357, -128.9316114714738 22.944844961737523, -128.7458949670052
25.77833126977869, -128.19192311448126 28.56333584149845, -127.27917452588864
31.252206476525537, -126.02326658263989 33.798935852798856, -124.44568821784746
36.159948724603595, -122.57343223472031 38.29484750710673, -120.43853345221717
40.16710349023388, -118.47656249999999 41.508577297439324)
Curnved WKT = CIRCULARSTRING (-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)
```

*Get control points from a Circular Ring*

```
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]
])
List<Point> points = circularRing.controlPoints
points.each { Point point ->
    println point
}
```

```
POINT (-118.47656249999999 41.508577297439324)
POINT (-109.6875 57.51582286553883)
POINT (-93.8671875 42.032974332441405)
POINT (-62.57812500000001 30.14512718337613)
POINT (-92.10937499999999 7.36246686553575)
POINT (-127.265625 14.604847155053898)
POINT (-118.47656249999999 41.508577297439324)
```

*Convert a Circural Ring to a Linear Geometry*

```
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]
])
Geometry linear = circularRing.linear
println linear.wkt
```

```
LINEARRING (-118.47656249999999 41.508577297439324, -118.58869123015452
41.901774579059115, -118.91479847512254 43.54122641035517, -119.02412442253558
45.209218040904574, -118.91479847512252 46.877209671453976, -118.58869123015451
48.516661502750026, -118.05138247300694 50.09952205941734, -117.31206570548474
51.59870815847947, -116.38339084245754 52.98856831012856, -115.28124776830906
54.245321621827856, -114.02449445660976 55.347464695976335, -112.63463430496067
56.27613955900354, -111.13544820589854 57.01545632652574, -109.6875 57.51582286553883,
-109.55258764923123 57.552765083673314, -107.91313581793517 57.87887232864133,
-106.24514418738576 57.98819827605437, -104.57715255683635 57.87887232864133,
-102.9377007255403 57.552765083673314, -101.35484016887298 57.01545632652574,
-99.85565406981085 56.27613955900354, -98.46579391816176 55.347464695976335,
-97.20904060646247 54.245321621827856, -96.10689753231398 52.98856831012856,
-95.17822266928678 51.59870815847947, -94.43890590176458 50.09952205941733,
-93.901597144617 48.51666150275002, -93.575489899649 46.87720967145396,
-93.46616395223595 45.20921804090456, -93.575489899649 43.54122641035516, -93.8671875
42.032974332441405, -92.52944589377746 42.87620255408929, -90.13998424943787
44.0545546346726, -87.61715897053445 44.91093841996455, -85.00413629704853
45.43070094596436, -82.34562577139025 45.60494893174677, -79.68711524573196
45.43070094596437, -77.07409257224604 44.910938419964566, -74.55126729334262
44.05455463467261, -72.16180564900301 42.876202554089296, -69.94659199044581
41.396044109014326, -67.9435292375422 39.639405220820365, -66.18689034934823
37.63634246791675, -64.70673190427325 35.42112880935955, -63.528379823689946
33.03166716501995, -62.67199603839799 30.50884188611652, -62.57812500000001
30.14512718337613, -62.15223351239818 27.895819212630613, -61.97798552661578
25.237308686972327, -62.15223351239818 22.578798161314037, -62.671996038397985
19.965775487828132, -63.52837982368993 17.442950208924703, -64.70673190427325
15.0534885645851, -66.18689034934822 12.838274906027902, -67.94352923754218
10.835212153124278, -69.9465919904458 9.078573264930323, -72.16180564900299
7.598414819855346, -74.55126729334259 6.42006273927203, -77.07409257224602
5.563678953980077, -79.68711524573193 5.0439164279802675, -82.34562577139022
4.869668442197863, -85.00413629704852 5.043916427980264, -87.61715897053442
5.5636789539800695, -90.13998424943784 6.420062739272023, -92.10937499999999
7.36246686553575, -94.00832592648507 5.722586433241059, -96.3693387982898
4.1450080684486466, -98.91606817456314 2.88910012519991, -101.60493880959022
1.9763515366073037, -104.38994338130999 1.4223796840833636, -107.22342968935115
1.2366631796147907, -110.05691599739232 1.4223796840833813, -112.84192056911208
1.9763515366073356, -115.53079120413916 2.8891001251999597, -118.07752058041248
4.1450080684487105, -120.43853345221721 5.722586433241133, -122.57343223472037
7.594842416368293, -124.4456882178475 9.729741198871439, -126.02326658263992
12.09075407067618, -127.265625 14.604847155053898, -127.27917452588866
14.637483446949503, -128.19192311448128 17.326354081976593, -128.7458949670052
20.111358653696357, -128.9316114714738 22.944844961737523, -128.7458949670052
25.77833126977869, -128.19192311448126 28.56333584149845, -127.27917452588864
31.252206476525537, -126.02326658263989 33.798935852798856, -124.44568821784746
36.159948724603595, -122.57343223472031 38.29484750710673, -120.43853345221717
40.16710349023388, -118.47656249999999 41.508577297439324)
```

# Compound Curves

*Get Curved WKT from a Compound Curves*

```
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]

        ])
])
println "WKT = ${compoundCurve.wkt}"
println "Curnved WKT = ${compoundCurve.curvedWkt}"
```

```
WKT = LINESTRING (27.0703125 23.885837699862005, 27.022331449414295 23.8488525605847,
25.524767337284196 22.84821221194479, 23.909405319245263 22.051603821364203,
22.203884687289108 21.472657579267334, 20.43738737228815 21.12127941637069,
18.64013863306764 21.00348151046186, 16.842889893847133 21.1212794163707,
15.076392578846175 21.472657579267356, 13.370871946890022 22.051603821364232,
11.755509928851094 22.848212211944833, 10.257945816720998 23.848852560584746,
8.903803347661494 25.036403633488835, 7.716252274757412 26.390546102548342,
6.715611926117504 27.88811021467844, 5.919003535536911 29.50347223271737,
5.340057293440038 31.20899286467353, 4.988679130543391 32.97549017967449,
4.8708812246345605 34.772738918894994, 4.988679130543396 36.5699876581155,
5.340057293440047 38.336484973116455, 5.9190035355369215 40.04200560507262, 5.9765625
40.17887331434696, 6.715611926117516 41.65736762311154, 7.71625227475743
43.15493173524164, 8.903803347661514 44.50907420430114, 10.257945816721021
45.69662527720523, 11.755509928851117 46.69726562584515, 13.37087194689005
47.49387401642574, 15.076392578846209 48.07282025852261, 16.84288989384717
48.42419842141926, 18.64013863306768 48.5419963273281, 20.437387372288192
48.42419842141926, 22.20388468728915 48.07282025852261, 22.5 47.98992166741417,
71.71875 49.15296965617039, 72.58658076138724 48.953451060440116, 74.12646468490568
48.43073090444654, 75.58494601643167 47.71148750685245, 76.93706973601537
46.80802732160263, 78.15970063193521 45.735808802953194, 79.23191915058464
44.513177907033366, 80.13537933583446 43.16105418744966, 80.85462273342854
41.70257285592367, 81.37734288942212 40.162688932405224, 81.5625 39.36827914916011,
81.69459591702075 38.567750257762206, 81.80095352896302 36.9450466749183,
81.69459591702075 35.3223430920744, 81.37734288942212 33.72740441743138,
80.85462273342854 32.187520493912935, 80.13537933583446 30.729039162386947,
79.23191915058464 29.376915442803238, 78.15970063193521 28.15428454688341,
76.93706973601537 27.082066028233974, 75.58494601643167 26.178605842984155,
74.12646468490568 25.459362445390067, 72.58658076138724 24.936642289396488,
70.99164208674422 24.61938926179787, 69.9609375 24.5271348225978)
Curnved WKT = COMPOUNDCURVE (CIRCULARSTRING (27.0703125 23.885837699862005, 5.9765625
40.17887331434696, 22.5 47.98992166741417), (22.5 47.98992166741417, 71.71875
49.15296965617039), CIRCULARSTRING (71.71875 49.15296965617039, 81.5625
39.36827914916011, 69.9609375 24.5271348225978))
```

*Get component LineStrings from a Compound Curves*

```
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]

        ])
])
List<LineString> lineStrings = compoundCurve.components
lineStrings.each { LineString lineString ->
    println lineString
}
```

```
LINESTRING (27.0703125 23.885837699862005, 27.022331449414295 23.8488525605847,
25.524767337284196 22.84821221194479, 23.909405319245263 22.051603821364203,
22.203884687289108 21.472657579267334, 20.43738737228815 21.12127941637069,
18.64013863306764 21.00348151046186, 16.842889893847133 21.1212794163707,
15.076392578846175 21.472657579267356, 13.370871946890022 22.051603821364232,
11.755509928851094 22.848212211944833, 10.257945816720998 23.848852560584746,
8.903803347661494 25.036403633488835, 7.716252274757412 26.390546102548342,
6.715611926117504 27.88811021467844, 5.919003535536911 29.50347223271737,
5.340057293440038 31.20899286467353, 4.988679130543391 32.97549017967449,
4.8708812246345605 34.772738918894994, 4.988679130543396 36.5699876581155,
5.340057293440047 38.336484973116455, 5.9190035355369215 40.04200560507262, 5.9765625
40.17887331434696, 6.715611926117516 41.65736762311154, 7.71625227475743
43.15493173524164, 8.903803347661514 44.50907420430114, 10.257945816721021
45.69662527720523, 11.755509928851117 46.69726562584515, 13.37087194689005
47.49387401642574, 15.076392578846209 48.07282025852261, 16.84288989384717
48.42419842141926, 18.64013863306768 48.5419963273281, 20.437387372288192
48.42419842141926, 22.20388468728915 48.07282025852261, 22.5 47.98992166741417)
LINESTRING (22.5 47.98992166741417, 71.71875 49.15296965617039)
LINESTRING (71.71875 49.15296965617039, 72.58658076138724 48.953451060440116,
74.12646468490568 48.43073090444654, 75.58494601643167 47.71148750685245,
76.93706973601537 46.80802732160263, 78.15970063193521 45.735808802953194,
79.23191915058464 44.513177907033366, 80.13537933583446 43.16105418744966,
80.85462273342854 41.70257285592367, 81.37734288942212 40.162688932405224, 81.5625
39.36827914916011, 81.69459591702075 38.567750257762206, 81.80095352896302
36.9450466749183, 81.69459591702075 35.3223430920744, 81.37734288942212
33.72740441743138, 80.85462273342854 32.187520493912935, 80.13537933583446
30.729039162386947, 79.23191915058464 29.376915442803238, 78.15970063193521
28.15428454688341, 76.93706973601537 27.082066028233974, 75.58494601643167
26.178605842984155, 74.12646468490568 25.459362445390067, 72.58658076138724
24.936642289396488, 70.99164208674422 24.61938926179787, 69.9609375 24.5271348225978)
```

*Convert a Compound Curves to a Linear Geometry*

```
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]

        ])
])
Geometry linear = compoundCurve.linear
println linear.wkt
```

```
LINESTRING (27.0703125 23.885837699862005, 27.022331449414295 23.8488525605847,
25.524767337284196 22.84821221194479, 23.909405319245263 22.051603821364203,
22.203884687289108 21.472657579267334, 20.43738737228815 21.12127941637069,
18.64013863306764 21.00348151046186, 16.842889893847133 21.1212794163707,
15.076392578846175 21.472657579267356, 13.370871946890022 22.051603821364232,
11.755509928851094 22.848212211944833, 10.257945816720998 23.848852560584746,
8.903803347661494 25.036403633488835, 7.716252274757412 26.390546102548342,
6.715611926117504 27.88811021467844, 5.919003535536911 29.50347223271737,
5.340057293440038 31.20899286467353, 4.988679130543391 32.97549017967449,
4.8708812246345605 34.772738918894994, 4.988679130543396 36.5699876581155,
5.340057293440047 38.336484973116455, 5.9190035355369215 40.04200560507262, 5.9765625
40.17887331434696, 6.715611926117516 41.65736762311154, 7.71625227475743
43.15493173524164, 8.903803347661514 44.50907420430114, 10.257945816721021
45.69662527720523, 11.755509928851117 46.69726562584515, 13.37087194689005
47.49387401642574, 15.076392578846209 48.07282025852261, 16.84288989384717
48.42419842141926, 18.64013863306768 48.5419963273281, 20.437387372288192
48.42419842141926, 22.20388468728915 48.07282025852261, 22.5 47.98992166741417,
71.71875 49.15296965617039, 72.58658076138724 48.953451060440116, 74.12646468490568
48.43073090444654, 75.58494601643167 47.71148750685245, 76.93706973601537
46.80802732160263, 78.15970063193521 45.735808802953194, 79.23191915058464
44.513177907033366, 80.13537933583446 43.16105418744966, 80.85462273342854
41.70257285592367, 81.37734288942212 40.162688932405224, 81.5625 39.36827914916011,
81.69459591702075 38.567750257762206, 81.80095352896302 36.9450466749183,
81.69459591702075 35.3223430920744, 81.37734288942212 33.72740441743138,
80.85462273342854 32.187520493912935, 80.13537933583446 30.729039162386947,
79.23191915058464 29.376915442803238, 78.15970063193521 28.15428454688341,
76.93706973601537 27.082066028233974, 75.58494601643167 26.178605842984155,
74.12646468490568 25.459362445390067, 72.58658076138724 24.936642289396488,
70.99164208674422 24.61938926179787, 69.9609375 24.5271348225978)
```

## Compound Rings

*Get Curved WKT from a Compound Rings*

```
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],
        ])
])
println "WKT = ${compoundRing.wkt}"
println "Curnved WKT = ${compoundRing.curvedWkt}"
```

```
WKT = LINEARRING (27.0703125 23.885837699862005, 27.022331449414295 23.8488525605847,
25.524767337284196 22.84821221194479, 23.909405319245263 22.051603821364203,
22.203884687289108 21.472657579267334, 20.43738737228815 21.12127941637069,
18.64013863306764 21.00348151046186, 16.842889893847133 21.1212794163707,
15.076392578846175 21.472657579267356, 13.370871946890022 22.051603821364232,
11.755509928851094 22.848212211944833, 10.257945816720998 23.848852560584746,
8.903803347661494 25.036403633488835, 7.716252274757412 26.390546102548342,
6.715611926117504 27.88811021467844, 5.919003535536911 29.50347223271737,
5.340057293440038 31.20899286467353, 4.988679130543391 32.97549017967449,
4.8708812246345605 34.772738918894994, 4.988679130543396 36.5699876581155,
5.340057293440047 38.336484973116455, 5.9190035355369215 40.04200560507262, 5.9765625
40.17887331434696, 6.715611926117516 41.65736762311154, 7.71625227475743
43.15493173524164, 8.903803347661514 44.50907420430114, 10.257945816721021
45.69662527720523, 11.755509928851117 46.69726562584515, 13.37087194689005
47.49387401642574, 15.076392578846209 48.07282025852261, 16.84288989384717
48.42419842141926, 18.64013863306768 48.5419963273281, 20.437387372288192
48.42419842141926, 22.20388468728915 48.07282025852261, 22.5 47.98992166741417,
71.71875 49.15296965617039, 72.58658076138724 48.953451060440116, 74.12646468490568
48.43073090444654, 75.58494601643167 47.71148750685245, 76.93706973601537
46.80802732160263, 78.15970063193521 45.735808802953194, 79.23191915058464
44.513177907033366, 80.13537933583446 43.16105418744966, 80.85462273342854
41.70257285592367, 81.37734288942212 40.162688932405224, 81.5625 39.36827914916011,
81.69459591702075 38.567750257762206, 81.80095352896302 36.9450466749183,
81.69459591702075 35.3223430920744, 81.37734288942212 33.72740441743138,
80.85462273342854 32.187520493912935, 80.13537933583446 30.729039162386947,
79.23191915058464 29.376915442803238, 78.15970063193521 28.15428454688341,
76.93706973601537 27.082066028233974, 75.58494601643167 26.178605842984155,
74.12646468490568 25.459362445390067, 72.58658076138724 24.936642289396488,
70.99164208674422 24.61938926179787, 69.9609375 24.5271348225978, 27.0703125
23.885837699862005)
Curnved WKT = COMPOUNDCURVE (CIRCULARSTRING (27.0703125 23.885837699862005, 5.9765625
40.17887331434696, 22.5 47.98992166741417), (22.5 47.98992166741417, 71.71875
49.15296965617039), CIRCULARSTRING (71.71875 49.15296965617039, 81.5625
39.36827914916011, 69.9609375 24.5271348225978), (69.9609375 24.5271348225978,
27.0703125 23.885837699862005))
```

*Get component LineStrings from a Compound Rings*

```
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],
        ])
])
List<LineString> lineStrings = compoundRing.components
lineStrings.each { LineString lineString ->
    println lineString
}
```

```
LINESTRING (27.0703125 23.885837699862005, 27.022331449414295 23.8488525605847,
25.524767337284196 22.84821221194479, 23.909405319245263 22.051603821364203,
22.203884687289108 21.472657579267334, 20.43738737228815 21.12127941637069,
18.64013863306764 21.00348151046186, 16.842889893847133 21.1212794163707,
15.076392578846175 21.472657579267356, 13.370871946890022 22.051603821364232,
11.755509928851094 22.848212211944833, 10.257945816720998 23.848852560584746,
8.903803347661494 25.036403633488835, 7.716252274757412 26.390546102548342,
6.715611926117504 27.88811021467844, 5.919003535536911 29.50347223271737,
5.340057293440038 31.20899286467353, 4.988679130543391 32.97549017967449,
4.8708812246345605 34.772738918894994, 4.988679130543396 36.5699876581155,
5.340057293440047 38.336484973116455, 5.9190035355369215 40.04200560507262, 5.9765625
40.17887331434696, 6.715611926117516 41.65736762311154, 7.71625227475743
43.15493173524164, 8.903803347661514 44.50907420430114, 10.257945816721021
45.69662527720523, 11.755509928851117 46.69726562584515, 13.37087194689005
47.49387401642574, 15.076392578846209 48.07282025852261, 16.84288989384717
48.42419842141926, 18.64013863306768 48.5419963273281, 20.437387372288192
48.42419842141926, 22.20388468728915 48.07282025852261, 22.5 47.98992166741417)
LINESTRING (22.5 47.98992166741417, 71.71875 49.15296965617039)
LINESTRING (71.71875 49.15296965617039, 72.58658076138724 48.953451060440116,
74.12646468490568 48.43073090444654, 75.58494601643167 47.71148750685245,
76.93706973601537 46.80802732160263, 78.15970063193521 45.735808802953194,
79.23191915058464 44.513177907033366, 80.13537933583446 43.16105418744966,
80.85462273342854 41.70257285592367, 81.37734288942212 40.162688932405224, 81.5625
39.36827914916011, 81.69459591702075 38.567750257762206, 81.80095352896302
36.9450466749183, 81.69459591702075 35.3223430920744, 81.37734288942212
33.72740441743138, 80.85462273342854 32.187520493912935, 80.13537933583446
30.729039162386947, 79.23191915058464 29.376915442803238, 78.15970063193521
28.15428454688341, 76.93706973601537 27.082066028233974, 75.58494601643167
26.178605842984155, 74.12646468490568 25.459362445390067, 72.58658076138724
24.936642289396488, 70.99164208674422 24.61938926179787, 69.9609375 24.5271348225978)
LINESTRING (69.9609375 24.5271348225978, 27.0703125 23.885837699862005)
```

*Convert a Compound Rings to a Linear Geometry*

```
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],
        ])
])
Geometry linear = compoundRing.linear
println linear.wkt
```

```
LINEARRING (27.0703125 23.885837699862005, 27.022331449414295 23.8488525605847,
25.524767337284196 22.84821221194479, 23.909405319245263 22.051603821364203,
22.203884687289108 21.472657579267334, 20.43738737228815 21.12127941637069,
18.64013863306764 21.00348151046186, 16.842889893847133 21.1212794163707,
15.076392578846175 21.472657579267356, 13.370871946890022 22.051603821364232,
11.755509928851094 22.848212211944833, 10.257945816720998 23.848852560584746,
8.903803347661494 25.036403633488835, 7.716252274757412 26.390546102548342,
6.715611926117504 27.88811021467844, 5.919003535536911 29.50347223271737,
5.340057293440038 31.20899286467353, 4.988679130543391 32.97549017967449,
4.8708812246345605 34.772738918894994, 4.988679130543396 36.5699876581155,
5.340057293440047 38.336484973116455, 5.9190035355369215 40.04200560507262, 5.9765625
40.17887331434696, 6.715611926117516 41.65736762311154, 7.71625227475743
43.15493173524164, 8.903803347661514 44.50907420430114, 10.257945816721021
45.69662527720523, 11.755509928851117 46.69726562584515, 13.37087194689005
47.49387401642574, 15.076392578846209 48.07282025852261, 16.84288989384717
48.42419842141926, 18.64013863306768 48.5419963273281, 20.437387372288192
48.42419842141926, 22.20388468728915 48.07282025852261, 22.5 47.98992166741417,
71.71875 49.15296965617039, 72.58658076138724 48.953451060440116, 74.12646468490568
48.43073090444654, 75.58494601643167 47.71148750685245, 76.93706973601537
46.80802732160263, 78.15970063193521 45.735808802953194, 79.23191915058464
44.513177907033366, 80.13537933583446 43.16105418744966, 80.85462273342854
41.70257285592367, 81.37734288942212 40.162688932405224, 81.5625 39.36827914916011,
81.69459591702075 38.567750257762206, 81.80095352896302 36.9450466749183,
81.69459591702075 35.3223430920744, 81.37734288942212 33.72740441743138,
80.85462273342854 32.187520493912935, 80.13537933583446 30.729039162386947,
79.23191915058464 29.376915442803238, 78.15970063193521 28.15428454688341,
76.93706973601537 27.082066028233974, 75.58494601643167 26.178605842984155,
74.12646468490568 25.459362445390067, 72.58658076138724 24.936642289396488,
70.99164208674422 24.61938926179787, 69.9609375 24.5271348225978, 27.0703125
23.885837699862005)
```

# Processing Geometries

*Get the geometry type (Point, LineString, Polygon, ect...) from a Geometry*

```
Geometry geom = Geometry.fromString("POINT (-124.80 48.92)")
String type = geom.geometryType
println type
```

```
Point
```

*Determine if one Geometry exactly equal another Geometry.*

```
Point point1 = new Point(-121.915, 47.390)
Point point2 = new Point(-121.915, 47.390)
Point point3 = new Point(-121.409, 47.413)

boolean does1equal2 = point1.equals(point2)
println "Does ${point1} equal ${point2}? ${does1equal2 ? 'Yes' : 'No'}"

boolean does1equal3 = point1.equals(point3)
println "Does ${point1} equal ${point3}? ${does1equal3 ? 'Yes' : 'No'}"

boolean does2equal3 = point2.equals(point3)
println "Does ${point2} equal ${point3}? ${does2equal3 ? 'Yes' : 'No'}"
```

```
Does POINT (-121.915 47.39) equal POINT (-121.915 47.39)? Yes
Does POINT (-121.915 47.39) equal POINT (-121.409 47.413)? No
Does POINT (-121.915 47.39) equal POINT (-121.409 47.413)? No
```

*Determine if one Geometry equals another Geometry topologically.*

```
Point point1 = new Point(-121.915, 47.390)
Point point2 = new Point(-121.915, 47.390)
Point point3 = new Point(-121.409, 47.413)

boolean does1equal2 = point1.equalsTopo(point2)
println "Does ${point1} equal ${point2}? ${does1equal2 ? 'Yes' : 'No'}"

boolean does1equal3 = point1.equalsTopo(point3)
println "Does ${point1} equal ${point3}? ${does1equal3 ? 'Yes' : 'No'}"

boolean does2equal3 = point2.equalsTopo(point3)
println "Does ${point2} equal ${point3}? ${does2equal3 ? 'Yes' : 'No'}"
```

```
Does POINT (-121.915 47.39) equal POINT (-121.915 47.39)? Yes
Does POINT (-121.915 47.39) equal POINT (-121.409 47.413)? No
Does POINT (-121.915 47.39) equal POINT (-121.409 47.413)? No
```

*Determine if one Geometry equals another Geometry when both are normalized.*

```
Geometry geom1 = Geometry.fromWKT("POLYGON ((2 4, 1 3, 2 1, 6 1, 6 3, 4 4, 2 4))")
Geometry geom2 = Geometry.fromWKT("POLYGON ((1 3, 2 4, 4 4, 6 3, 6 1, 2 1, 1 3))")
Geometry geom3 = Geometry.fromWKT("POLYGON ((1 1, 1 4, 4 4, 4 1, 1 1))")

boolean does1equal2 = geom1.equalsNorm(geom2)
println "Does ${geom1} equal ${geom2}? ${does1equal2 ? 'Yes' : 'No'}"

boolean does1equal3 = geom1.equalsNorm(geom3)
println "Does ${geom1} equal ${geom3}? ${does1equal3 ? 'Yes' : 'No'}"

boolean does2equal3 = geom2.equalsNorm(geom3)
println "Does ${geom2} equal ${geom3}? ${does2equal3 ? 'Yes' : 'No'}"
```

```
Does POLYGON ((2 4, 1 3, 2 1, 6 1, 6 3, 4 4, 2 4)) equal POLYGON ((1 3, 2 4, 4 4, 6 3,
6 1, 2 1, 1 3))? Yes
Does POLYGON ((2 4, 1 3, 2 1, 6 1, 6 3, 4 4, 2 4)) equal POLYGON ((1 1, 1 4, 4 4, 4 1,
1 1))? No
Does POLYGON ((1 3, 2 4, 4 4, 6 3, 6 1, 2 1, 1 3)) equal POLYGON ((1 1, 1 4, 4 4, 4 1,
1 1))? No
```

*Get a Geometry by index in a GeometryCollection*

```
MultiPoint multiPoint = new MultiPoint([
        new Point(-122.3876953125, 47.5820839916191),
        new Point(-122.464599609375, 47.25686404408872),
        new Point(-122.48382568359374, 47.431803338643334)
])
Point p1 = multiPoint[0]
println p1

Point p2 = multiPoint[1]
println p2

Point p3 = multiPoint[2]
println p3
```

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

*Cast a Geometry to a Bounds or to a Point*

```
Geometry geometry = 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]
]])

Bounds bounds = geometry as Bounds
println bounds

Point point = geometry as Point
println point
```

```
(-122.64,46.308,-120.981,47.413)
POINT (-121.73789467295867 46.95085967283822)
```

*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
```

*Check whether a Geometry is within another Geometry*

```
Polygon polygon1 = new Polygon([[
        [-120.212, 47.591],
        [-119.663, 47.591],
        [-119.663, 47.872],
        [-120.212, 47.872],
        [-120.212, 47.591]
]])

Polygon polygon2 = 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]
]])

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

```
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]
]])

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

```
false
```

*Check whether a Geometry touches another Geometry*

```
LineString line1 = new LineString([
    [-122.38651514053345, 47.58219978280006],
    [-122.38651514053345, 47.58020234903306]
])

LineString line2 = new LineString([
    [-122.38651514053345, 47.58124449789785],
    [-122.38333940505981, 47.58124449789785]
])

boolean touches = line1.touches(line2)
```

```
true
```

```
LineString line3 = new LineString([
    [-122.386257648468, 47.58183793450921],
    [-122.38348960876465, 47.5818668824645]
])

touches = line1.touches(line3)
```

```
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)
```

*Calculate the symmetric 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 symDifference = polygon1.symDifference(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)
```

*Perform a cascaded union on a List of Polygons.*

```
Geometry randomPoints = Geometry.createRandomPoints(new Bounds(-180,-90,180,90,
"EPSG:4326").geometry, 100)
List<Geometry> bufferedPoints = randomPoints.collect{Geometry geom -> geom.buffer(
4.5)}
Geometry unionedGeometry = Geometry.cascadedUnion(bufferedPoints)
```

*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)
```

*Create a delaunay triangle diagram around a Geometry*

```
Geometry points = Geometry.createRandomPoints(new Bounds(-180, -90, 180, 90).geometry,
100)
Geometry delaunayTriangle = points.delaunayTriangleDiagram
```

*Create a voronoi diagram around a Geometry*

```
Geometry points = Geometry.createRandomPoints(new Bounds(-180, -90, 180, 90).geometry,
100)
Geometry voronoiDiagram = points.voronoiDiagram
```

*Calculate the union of 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 union = polygon1.union(polygon2)
```

*Calculate the intersection 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 intersection = polygon1.intersection(polygon2)
```

*Check whether one Geometry intersects 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 does1intersect2 = polygon1.intersects(polygon2)
println does1intersect2

boolean does1intersect3 = polygon1.intersects(polygon3)
println does1intersect3

boolean does2intersect3 = polygon2.intersects(polygon3)
println does2intersect3
```

```
true
false
false
```

*Check whether one Geometry overlaps 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 does1overlap2 = polygon1.overlaps(polygon2)
println does1overlap2

boolean does1overlap3 = polygon1.overlaps(polygon3)
println does1overlap3

boolean does2overlap3 = polygon2.overlaps(polygon3)
println does2overlap3
```

```
true
false
false
```

*Calculate the octagonal envelope of a Geometry*

```
Geometry geometry = 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]
]])
Geometry octagonalEnvelope = geometry.octagonalEnvelope
```

*Calculate the minimum rectangle of a Geometry*

```
Geometry geometry = 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]
]])
Geometry minimumRectangle = geometry.minimumRectangle
```

*Calculate the minimum circle of a Geometry*

```
Geometry geometry = 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]
]])
Geometry minimumBoundingCircle = geometry.minimumBoundingCircle
```

*Calculate the minimum diameter of a Geometry*

```
Geometry geometry = 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]
]])
Geometry minimumDiameter = geometry.minimumDiameter
```

*Calculate the minimum clearance of a Geometry*

```
Geometry geometry = 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]
]])
Geometry minimumClearance = geometry.minimumClearance
```

*Offset a LineString by a given distance. Positive distances will offset to the right. Negative distance will offset to the left.*

```
LineString line = new LineString(
        [3.198, 43.164],
        [6.713, 49.755],
        [9.702, 42.592],
        [15.32, 53.798]
)
LineString positive = line.offset(1.2)
LineString negative = line.offset(-2.4)
```

*Get the dimension of a Geometry*

```
Point point = Geometry.fromWKT("POINT (-122.3437 47.7540)")
println "Point Dimension = ${point.dimension}"

LineString lineString = Geometry.fromWKT("LINESTRING (-122.525 47.256, -122.376
47.595)")
println "LineString Dimension = ${lineString.dimension}"

Polygon polygon = Geometry.fromWKT("POLYGON ((-122.590 47.204, -122.365 47.204,
-122.365 47.312, -122.590 47.312, -122.590 47.204))")
println "Polygon Dimension = ${polygon.dimension}"
```

```
Point Dimension = 0
LineString Dimension = 1
Polygon Dimension = 2
```

*Determine if a Geometry is empty or not*

```
Geometry geom1 = Geometry.fromWKT("POINT EMPTY")
boolean isGeom1Empty = geom1.empty
println "Is ${geom1.wkt} empty? ${isGeom1Empty ? 'Yes' : 'No'}"

Geometry geom2 = Geometry.fromWKT("POINT (-122.3437 47.7540)")
boolean isGeom2Empty = geom2.empty
println "Is ${geom2.wkt} empty? ${isGeom2Empty ? 'Yes' : 'No'}"
```

```
Is POINT EMPTY empty? Yes
Is POINT (-122.3437 47.754) empty? No
```

*Determine if a Geometry is rectanglular*

```
Geometry geom1 = Geometry.fromWKT("POLYGON ((-122.590 47.204, -122.365 47.204,
-122.365 47.312, -122.590 47.312, -122.590 47.204))")
boolean isGeom1Rect = geom1.isRectangle()
println "Is the geometry a rectangle? ${isGeom1Rect ? 'Yes' : 'No'}"
```

```
Is the geometry a rectangle? Yes
```

```
Geometry geom2 = Geometry.fromWKT("POLYGON ((-122.360 47.215, -122.656 46.912,
-121.838 46.931, -122.360 47.215))")
boolean isGeom2Rect = geom2.isRectangle()
println "Is the geometry a rectangle? ${isGeom2Rect ? 'Yes' : 'No'}"
```

```
Is the geometry a rectangle? No
```

*Determine if a Geometry is simple*

```
Geometry geom1 = new LineString(
        [-122.323, 47.599],
        [-122.385, 47.581]
)
boolean isGeom1Simple = geom1.simple
println "Is the Geometry simple? ${isGeom1Simple}"
```

```
Is the Geometry simple? true
```

```
Geometry geom2 = new LineString(
        [-122.356, 47.537],
        [-122.295, 47.580],
        [-122.284, 47.532],
        [-122.353, 47.574]
)
boolean isGeom2Simple = geom2.simple
println "Is the Geometry simple? ${isGeom2Simple}"
```

```
Is the Geometry simple? false
```

*Determine if a Geometry is valid*

```
Geometry geom1 = new LineString(
        [-122.323, 47.599],
        [-122.385, 47.581]
)
boolean isGeom1Valid = geom1.valid
println "Is the Geometry valid? ${isGeom1Valid}"
```

```
Is the Geometry valid? true
```

```
Geometry geom2 = new Polygon(new LinearRing([
    [48.16406, 42.29356],
    [35.15625, 25.79989],
    [64.33593, 24.52713],
    [26.71875, 39.09596],
    [48.16406, 42.29356],
]))
boolean isGeom2Valid = geom2.valid
println "Is the Geometry valid? ${isGeom2Valid}"
println geom2.validReason
```

```
Is the Geometry valid? false
Self-intersection
```

*Determine if a Geometry is curved*

```
Geometry geom1 = new CircularString([
        [-122.464599609375, 47.247542522268006],
        [-122.03613281249999, 47.37789454155521],
        [-122.37670898437499, 47.58393661978134]
])

boolean isGeom1Curved = geom1.curved
println "Is the Geometry valid? ${isGeom1Curved}"
```

```
Is the Geometry curved? true
```

```
Geometry geom2 = new LineString(
        [-122.323, 47.599],
        [-122.385, 47.581]
)
boolean isGeom2Curved = geom2.curved
println "Is the Geometry valid? ${isGeom2Curved}"
```

```
Is the Geometry curved? false
```

*Determine if a Geometry is within a given distance of another Geometry*

```
Geometry geom1 = new Point(-88.945, 41.771)
Geometry geom2 = new Point(-113.906, 37.160)

double distance1 = 26.0
boolean isWithin1 = geom1.isWithinDistance(geom2, distance1)
println "Is ${geom1} within ${distance1} of ${geom2}? ${isWithin1 ? 'Yes' : 'No'}"
```

```
Is POINT (-88.945 41.771) within 26.0 of POINT (-113.906 37.16)? Yes
```

```
double distance2 = 15.5
boolean isWithin2 = geom1.isWithinDistance(geom2, distance2)
println "Is ${geom1} within ${distance2} of ${geom2}? ${isWithin2 ? 'Yes' : 'No'}"
```

```
Is POINT (-88.945 41.771) within 15.5 of POINT (-113.906 37.16)? No
```

*Normalizing a Geometry changes the Geometry in place.*

```
Geometry geometry = Geometry.fromWKT("POLYGON((2 4, 1 3, 2 1, 6 1, 6 3, 4 4, 2 4))")
geometry.normalize()
println "Normalized Geometry = ${geometry}"
```

```
Normalized Geometry = POLYGON ((1 3, 2 4, 4 4, 6 3, 6 1, 2 1, 1 3))
```

*Calculating a normalized Geometry from a Geometry does not change the original Geometry.*

```
Geometry geometry = Geometry.fromWKT("POLYGON((2 4, 1 3, 2 1, 6 1, 6 3, 4 4, 2 4))")
Geometry normalizedGeometry = geometry.norm
println "Un-normalized Geometry = ${geometry}"
println "Normalized Geometry    = ${normalizedGeometry}"
```

```
Un-normalized Geometry = POLYGON ((2 4, 1 3, 2 1, 6 1, 6 3, 4 4, 2 4))
Normalized Geometry = POLYGON ((1 3, 2 4, 4 4, 6 3, 6 1, 2 1, 1 3))
```

*Smooth a Geometry*

```
Geometry geometry = Geometry.fromWKT("POLYGON((10 0, 10 20, 0 20, 0 30, 30 30, 30 20,
20 20, 20 0, 10 0))")
Geometry smoothed = geometry.smooth(0.75)
```

*Calculate the DE-9IM Intersection Matrix 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]
]])

IntersectionMatrix matrix = polygon1.relate(polygon2)
println "Intersection Matrix = ${matrix}"
println "Contains = ${matrix.contains}"
println "Covered By = ${matrix.coveredBy}"
println "Covers = ${matrix.covers}"
println "Disjoint = ${matrix.disjoint}"
println "Intersects = ${matrix.intersects}"
println "Within = ${matrix.within}"
```

```
Intersection Matrix = 212101212
Contains = false
Covered By = false
Covers = false
Disjoint = false
Intersects = true
Within = false
```

*Determine if a Geometry relates to another Geometry according to the given DE-9IM Intersection Matrix string*

```
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]
]])

println polygon1.relate(polygon2, "212101212")
println polygon1.relate(polygon2, "111111111")
println polygon1.relate(polygon2, "222222222")
```

```
true
false
false
```

*Densify a Geometry by adding points at a given interval.*

```
Geometry geometry = new LineString([
    [-122.28062152862547, 47.12986316579223],
    [-122.2809863090515, 47.12935221617075],
    [-122.2809863090515, 47.12786313499169],
    [-122.28111505508421, 47.127731743474406],
    [-122.28137254714966, 47.127673347140345],
    [-122.28178024291992, 47.12768794622986],
    [-122.28227376937865, 47.128067521151195],
    [-122.28227376937865, 47.12906024275466]
])
Geometry densified = geometry.densify(0.0001)
println "# of points in original geometry = ${geometry.numPoints}"
println "# of points in densified geometry = ${densified.numPoints}"
```

```
# of points in original geometry = 8
# of points in densified geometry = 50
```

*Simplify a Geometry by removing points at a given interval.*

```
Geometry geometry = new LineString([
    [-123.59619140625001, 47.338822694822],
    [-123.04687499999999, 47.010225655683485],
    [-122.2119140625, 46.965259400349275],
    [-121.201171875, 47.17477833929903],
    [-120.87158203125, 47.487513008956554],
    [-120.62988281249999, 48.31242790407178],
    [-120.84960937499999, 48.647427805533546],
    [-121.59667968749999, 48.850258199721495],
    [-122.36572265625, 48.980216985374994],
    [-123.134765625, 48.83579746243093],
    [-123.3984375, 48.44377831058802],
    [-123.59619140625001, 48.10743118848039],
    [-123.85986328124999, 47.62097541515849]
])
Geometry simplified = geometry.simplify(0.5)
println "# of points in original geometry = ${geometry.numPoints}"
println "# of points in simplified geometry = ${simplified.numPoints}"
```

```
# of points in original geometry = 13
# of points in simplified geometry = 5
```

*Simplify a Geometry by removing points at a given interval but preserve the topological structure.*

```
Geometry geometry = new LineString([
        [-123.59619140625001, 47.338822694822],
        [-123.04687499999999, 47.010225655683485],
        [-122.2119140625, 46.965259400349275],
        [-121.201171875, 47.17477833929903],
        [-120.87158203125, 47.487513008956554],
        [-120.62988281249999, 48.31242790407178],
        [-120.84960937499999, 48.647427805533546],
        [-121.59667968749999, 48.850258199721495],
        [-122.36572265625, 48.980216985374994],
        [-123.134765625, 48.83579746243093],
        [-123.3984375, 48.44377831058802],
        [-123.59619140625001, 48.10743118848039],
        [-123.85986328124999, 47.62097541515849]
])
Geometry simplified = geometry.simplifyPreservingTopology(0.1)
println "# of points in original geometry = ${geometry.numPoints}"
println "# of points in simplified geometry = ${simplified.numPoints}"
```

```
# of points in original geometry = 13
# of points in simplified geometry = 11
```

*Translate or move a geometry a given distance along the x and y axis.*

```
Geometry geometry = new Polygon(new LinearRing([
    [-121.83837890625, 47.5913464767971],
    [-122.76123046875, 46.9802523552188],
    [-122.67333984374, 46.3014061543733],
    [-121.00341796874, 46.3772542051002],
    [-121.22314453124, 47.1448974855539],
    [-121.83837890625, 47.5913464767971]
]))
Geometry translatedGeometry = geometry.translate(2.1, 3.2)
```

*Scale a geometry a given amount in an x and y direction around the origin*

```
Geometry geometry = new Polygon(new LinearRing([
        [-121.83837890625, 47.5913464767971],
        [-122.76123046875, 46.9802523552188],
        [-122.67333984374, 46.3014061543733],
        [-121.00341796874, 46.3772542051002],
        [-121.22314453124, 47.1448974855539],
        [-121.83837890625, 47.5913464767971]
]))
Geometry scaledGeometry = geometry.scale(1.1,1.2)
println scaledGeometry
```

*Scale a geometry a given amount in an x and y direction around a point*

```
Geometry geometry = new Polygon(new LinearRing([
        [-121.83837890625, 47.5913464767971],
        [-122.76123046875, 46.9802523552188],
        [-122.67333984374, 46.3014061543733],
        [-121.00341796874, 46.3772542051002],
        [-121.22314453124, 47.1448974855539],
        [-121.83837890625, 47.5913464767971]
]))
Point centroid = geometry.centroid
Geometry scaledGeometry = geometry.scale(1.1, 1.1, centroid.x, centroid.y)
```

*Rotate a Geometry around it's origin by a given angle theta (in radians).*

```
Geometry geometry = new Polygon(new LinearRing([
        [-121.83837890625, 47.5913464767971],
        [-122.76123046875, 46.9802523552188],
        [-122.67333984374, 46.3014061543733],
        [-121.00341796874, 46.3772542051002],
        [-121.22314453124, 47.1448974855539],
        [-121.83837890625, 47.5913464767971]
]))
Geometry theta = geometry.rotate(Math.toRadians(45))
```

*Rotate a Geometry around an XY coordinate by a given angle theta (in radians).*

```
Geometry thetaXY = geometry.rotate(Math.toRadians(90), geometry.centroid.x, geometry
.centroid.y)
```

*Rotate a Geometry around it's origin by a given angle sine and cosine (in radians).*

```
Geometry sinCos = geometry.rotate(Math.toRadians(15), Math.toRadians(35))
```

*Rotate a Geometry around an XY coordinate by a given angle sine and cosine (in radians).*

```
Geometry sinCosXY = geometry.rotate(Math.toRadians(15), Math.toRadians(35), geometry
.centroid.x, geometry.centroid.y)
```

*Shear a Geometry around it's origin by a given distance along the x and y axis.*

```
Geometry geometry = new Polygon(new LinearRing([
        [-121.83837890625, 47.5913464767971],
        [-122.76123046875, 46.9802523552188],
        [-122.67333984374, 46.3014061543733],
        [-121.00341796874, 46.3772542051002],
        [-121.22314453124, 47.1448974855539],
        [-121.83837890625, 47.5913464767971]
]))
Geometry shearedGeometry = geometry.shear(0.1,0.4)
```

*Reflect a Geometry around an XY coordinate for given distance along the x and y axis*

```
Geometry geometry = new Polygon(new LinearRing([
        [-121.83837890625, 47.5913464767971],
        [-122.76123046875, 46.9802523552188],
        [-122.67333984374, 46.3014061543733],
        [-121.00341796874, 46.3772542051002],
        [-121.22314453124, 47.1448974855539],
        [-121.83837890625, 47.5913464767971]
]))
Point centroid = geometry.centroid
Geometry reflectedGeometry = geometry.reflect(0.1,0.4, centroid.x, centroid.y)
```

*Reflect a Geometry around the origin for given distance along the x and y axis*

```
Geometry reflectedAroundOrigin = geometry.reflect(0.5, 0.34)
```

*Reduce the precision of a Geometry's coordinates.*

```
Geometry g1 = new Point(5.19775390625, 51.07421875)
println "Original Geometry: ${g1.wkt}"

Geometry g2 = g1.reducePrecision()
println "Floating Point Geometry: ${g2.wkt}"

Geometry g3 = g1.reducePrecision("fixed", scale: 100)
println "Fixed Point Geometry: ${g3.wkt}"

Geometry g4 = g1.reducePrecision("floating_single", pointwise: true, removecollapsed:
true)
println "Floating Point Single Geometry: ${g4.wkt}"
```

```
Original Geometry: POINT (5.19775390625 51.07421875)
Floating Point Geometry: POINT (5.19775390625 51.07421875)
Fixed Point Geometry: POINT (5.2 51.07)
Floating Point Single Geometry: POINT (5.19775390625 51.07421875)
```

*Find the nearest points in one Geometry to another*

```
Geometry point = new Point( -122.3845276236534, 47.58285653105873)
Geometry polygon = Geometry.fromWKT("POLYGON ((-122.3848307132721 47.58285110342092, "
+
        "-122.38484144210814 47.58255620092149, -122.38469392061235
47.582558010144346, " +
        "-122.3846912384033 47.5825797208137, -122.38460808992384 47.58258695770149, "
+
        "-122.38460808992384 47.582628569786834, -122.38458126783371
47.58263037900717, " +
        "-122.38458126783371 47.58277330721735, -122.38460540771483 47.58277149800195,
" +
        "-122.38460540771483 47.582805873084084, -122.38467246294022 47.5828131099406,
" +
        "-122.38467246294022 47.58285110342092, -122.3848307132721
47.58285110342092))")
List<Point> nearestPoints = polygon.getNearestPoints(point)
```

*Convert a Geometry to a PreparedGeometry for more effecient spatial queries.*

```
Geometry geometry = 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]
]])
PreparedGeometry preparedGeometry = geometry.prepare()

Closure timer = { Closure action ->
    long start = System.nanoTime()
    action.call()
    long end = System.nanoTime()
    end - start
}

MultiPoint points = Geometry.createRandomPoints(new Bounds(-180, -90, 180, 90
).geometry, 100000)


long timeWithGeometry = timer({ ->
    points.geometries.each { Point point ->
        geometry.contains(point)
    }
})
println "Time with Geometry         = ${timeWithGeometry} nanoseconds"

long timeWithPreparedGeometry = timer({ ->
    points.geometries.each { Point point ->
        preparedGeometry.contains(point)
    }
})

println "Time with PreparedGeometry = ${timeWithPreparedGeometry} nanoseconds"
```

```
Time with Geometry         = 269595687 nanoseconds
Time with PreparedGeometry = 197789535 nanoseconds
```

*Convert a Geometry to a PreparedGeometry using a static method for more effecient spatial queries.*

```
Geometry geometry = 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]
]])
PreparedGeometry preparedGeometry = Geometry.prepare(geometry)

Closure timer = { Closure action ->
    long start = System.nanoTime()
    action.call()
    long end = System.nanoTime()
    end - start
}

MultiPoint points = Geometry.createRandomPoints(new Bounds(-180, -90, 180, 90
).geometry, 100000)


long timeWithGeometry = timer({ ->
    points.geometries.each { Point point ->
        geometry.contains(point)
    }
})
println "Time with Geometry         = ${timeWithGeometry} nanoseconds"

long timeWithPreparedGeometry = timer({ ->
    points.geometries.each { Point point ->
        preparedGeometry.contains(point)
    }
})

println "Time with PreparedGeometry = ${timeWithPreparedGeometry} nanoseconds"
```

```
Time with Geometry         = 192898879 nanoseconds
Time with PreparedGeometry = 134561246 nanoseconds
```

# Prepared Geometry

PreparedGeometry are more effecient for spatial queries.

*You can create a PreparedGeometry by wrapping an existing Geometry. You can then get the original Geometry with the geometry property.*

```
Polygon polygon = 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]
]])

PreparedGeometry preparedGeometry = new PreparedGeometry(polygon)

Geometry geometry = preparedGeometry.geometry
```

*Check whether a PreparedGeometry contains another Geometry*

```
PreparedGeometry polygon1 = new PreparedGeometry(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
```

```
false
```

*Check whether a PreparedGeometry properly contains another Geometry. This means that every point in the other geometry doesn't intersect with the first.*

```
PreparedGeometry polygon1 = new PreparedGeometry(new Polygon([[
    [-122.50064849853516, 47.22096718353454],
    [-122.41928100585938, 47.22096718353454],
    [-122.41928100585938, 47.277365616965646],
    [-122.50064849853516, 47.277365616965646],
    [-122.50064849853516, 47.22096718353454]
]]))

Polygon polygon2 = new Polygon([[
    [-122.44571685791014, 47.24031721435104],
    [-122.42958068847656, 47.24031721435104],
    [-122.42958068847656, 47.253135632244216],
    [-122.44571685791014, 47.253135632244216],
    [-122.44571685791014, 47.24031721435104]
]])

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

```
true
```

```
false
```

*Check whether a PreparedGeometry 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]
]])

PreparedGeometry polygon2 = new PreparedGeometry(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
```

```
false
```

*Check whether a PreparedGeometry covers another Geometry.*

```
PreparedGeometry polygon1 = new PreparedGeometry(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
```

```
false
```

*Check whether a PreparedGeometry crosses another Geometry.*

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

boolean doesCross12 = line1.crosses(line2)
println doesCross12

boolean doesCross13 = line1.crosses(line3)
println doesCross13
```

```
true
false
```

*Check whether a PreparedGeometry is disjoint from another Geometry.*

```
PreparedGeometry polygon1 = new PreparedGeometry(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
```

```
false
true
```

*Check whether a PreparedGeometry intersects another Geometry.*

```
PreparedGeometry polygon1 = new PreparedGeometry(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 does1intersect2 = polygon1.intersects(polygon2)
println does1intersect2

boolean does1intersect3 = polygon1.intersects(polygon3)
println does1intersect3
```

```
true
false
```

*Check whether a PreparedGeometry overlaps another Geometry.*

```
PreparedGeometry polygon1 = new PreparedGeometry(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 does1overlap2 = polygon1.overlaps(polygon2)
println does1overlap2

boolean does1overlap3 = polygon1.overlaps(polygon3)
println does1overlap3
```

```
true
false
```

*Check whether a PreparedGeometry touches another Geometry.*

```
PreparedGeometry line1 = new PreparedGeometry(new LineString([
    [-122.38651514053345, 47.58219978280006],
    [-122.38651514053345, 47.58020234903306]
]))

LineString line2 = new LineString([
    [-122.38651514053345, 47.58124449789785],
    [-122.38333940505981, 47.58124449789785]
])

boolean touches = line1.touches(line2)
```

```
true
```

```
LineString line3 = new LineString([
    [-122.386257648468, 47.58183793450921],
    [-122.38348960876465, 47.5818668824645]
])

touches = line1.touches(line3)
```

```
false
```

*Check whether a PreparedGeometry is within another Geometry.*

```
PreparedGeometry polygon1 = new PreparedGeometry(new Polygon([[
    [-120.212, 47.591],
    [-119.663, 47.591],
    [-119.663, 47.872],
    [-120.212, 47.872],
    [-120.212, 47.591]
]]))

Polygon polygon2 = 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]
]])

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

```
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]
]])

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

```
false
```

# 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
GeoPackageReader
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
GeoPackageWriter
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]}
```

*Create a Geometry from a String. The string will be parse by each Geometry Reader.*

```
Geometry geom1 = Geometry.fromString('POINT (-123.15 46.237)')
println geom1

Geometry geom2 = Geometry.fromString
('{"type":"LineString","coordinates":[[3.198,43.164],[6.713,49.755],[9.702,42.592],[15
.32,53.798]]}')
println geom2


Geometry geom3 = Geometry.fromString('<Point><coordinates>-
123.15,46.237</coordinates></Point>')
println geom3
```

```
POINT (-123.15 46.237)
LINESTRING (3.198 43.164, 6.713 49.755, 9.702 42.592, 15.32 53.798)
POINT (-123.15 46.237)
```

## 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
```

```
0000000002000000044009958106 24DD2F404594FDF3B645A2401ADA1CAC0831274048E0A3D70A3D714023
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
```

```
(1147444.7684517875,703506.2231641772,1155828.1202425088,711367.9403610165,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 703068.0661327888)
```

*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 703068.0661327888)
```

*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 703068.0661327888)
```

*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 703068.0661327888)
```

## 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
```

```
75.6581745719509
```

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

```
4164050.459880065
```

*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 43.52791689304305)
POINT (-79.8828640042499 44.63747566950249)
POINT (-84.51118758826816 45.565540142641)
POINT (-89.27793446221685 46.300124344169255)
POINT (-94.15564606698499 46.83102721803566)
POINT (-99.11079892605703 47.15045006457598)
POINT (-104.10532353179985 47.25351783423774)
POINT (-109.09873812691619 47.13862709798195)
POINT (-114.05062990603696 46.80756425557423)
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)
```

*Create a scatter plot chart with options*

```
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, legend: false, xLabel: "X Coordinates",
yLabel: "Y Coordinates")
```

# 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-2114f051_168da228c02_-2aea 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
xmlns:georss='http://www.georss.org/georss'
xmlns='http://www.w3.org/2005/Atom'>[geom:POINT (-122.3204 47.6024), id:1,
name:Seattle]</summary><updated>Mon Feb 11 01:20:09 UTC 2019</updated><georss:point
xmlns:georss='http://www.georss.org/georss'
xmlns='http://www.w3.org/2005/Atom'>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-2114f051_168da228c02_-2aee 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
xmlns:georss='http://www.georss.org/georss'
xmlns='http://www.w3.org/2005/Atom'>[geom:POINT (-122.3204 47.6024), id:1,
name:Seattle]</summary><updated>Mon Feb 11 01:20:09 UTC 2019</updated><georss:point
xmlns:georss='http://www.georss.org/georss'
xmlns='http://www.w3.org/2005/Atom'>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-2114f051_168da228c02_-2aec 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-2114f051_168da228c02_-2aeb 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-2114f051_168da228c02_-2aed 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 301 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
```

# Process Functions

Process Functions are a combination of Functions and Processes that can be used to create rendering transformations.

*Create a Function from a Process that converts geometries in a Layer into a convexhull.*

```
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
```

*Create a ProcessFunction from a Process that converts geometries in a Layer into a bounds.*

```
Workspace workspace = new GeoPackage('src/main/resources/data.gpkg')
Layer places = workspace.get("places")
Process process = new Process("bounds",
        "Create a bounds 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.bounds.geometry])
            [result: output]
        }
)
ProcessFunction processFunction = new ProcessFunction(process, new Function
("parameter", new Expression("features")))
Symbolizer symbolizer =  new Transform(processFunction, Transform.RENDERING) + new
Fill("aqua", 0.75) + new Stroke("navy", 0.5)
places.style = symbolizer
```

## 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])
```

# Create Expression from static imports

*You can import short helper methods from the Expressions class.*

```
import static geoscript.filter.Expressions.*
```

*Create a literal*

```
Expression literal = expression(1.2)
```

*Create a Color*

```
Expression color = color("wheat")
```

*Create a Property*

```
Expression property = property("ID")
```

*Create a Function*

```
Expression function = function("max(10,22)")
```

# 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:ClassifyByRange
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:getX
geo:getY
geo:isValid
geo:buffer
geo:union
geo:intersection
geo:difference
geo:reproject
geo:splitPolygon
```

```
geo:within
geo:touches
geo:convexHull
geo:area
geo:crosses
geo:symDifference
geo:distance
geo:boundary
geo:centroid
geo:interiorPoint
geo:getGeometryN
geo:overlaps
geo:isSimple
geo:isWithinDistance
geo:relate
geo:densify
geo:simplify
geo:numGeometries
geo:startPoint
geo:numPoints
geo:isClosed
geo:dimension
geo:exteriorRing
geo:numInteriorRing
geo:geometryType
geo:envelope
geo:isRing
geo:polygonize
geo:endPoint
geo:equalsExact
geo:pointN
geo:relatePattern
geo:interiorRingN
geo:equalsExactTolerance
geo:length
geo:isEmpty
geo:contains
geo:disjoint
geo:intersects
polygonlabelprocess:PolyLabeller
ras:AddCoverages
ras:Affine
ras:AreaGrid
ras:BandMerge
ras:BandSelect
ras:Contour
ras:ConvolveCoverage
ras:CoverageClassStats
ras:CropCoverage
ras:Jiffle
ras:MultiplyCoverages
```

```
ras:NormalizeCoverage
ras:PolygonExtraction
ras:RangeLookup
ras:RasterAsPointCollection
ras:RasterZonalStatistics
ras:RasterZonalStatistics2
ras:ScaleCoverage
ras:StyleCoverage
ras:TransparencyFill
geoscript:convexhull
```

# 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
```

# Process Functions

Process Functions are a combination of Functions and Processes that can be used to create rendering transformations.

*Create a Function from a Process that converts geometries in a Layer into a convexhull.*

```
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
```

*Create a ProcessFunction from a Process that converts geometries in a Layer into a bounds.*

```
Workspace workspace = new GeoPackage('src/main/resources/data.gpkg')
Layer places = workspace.get("places")
Process process = new Process("bounds",
        "Create a bounds 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.bounds.geometry])
            [result: output]
        }
)
ProcessFunction processFunction = new ProcessFunction(process, new Function
("parameter", new Expression("features")))
Symbolizer symbolizer =  new Transform(processFunction, Transform.RENDERING) + new
Fill("aqua", 0.75) + new Stroke("navy", 0.5)
places.style = symbolizer
```

# Render Recipes

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

## Creating Maps

*Create a Map with Layers and render to a File.*

```
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)
```

*Create a Map with Layers and render to a file name.*

```
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]
)
map.render("map.png")
```

*Create a Map with Layers and render to an OutputStream.*

```
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")
file.withOutputStream { OutputStream outputStream ->
    map.render(outputStream)
}
```

*Create a Map with Layers and render to an BufferedImage.*

```
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]
)
BufferedImage image = map.renderToImage()
```

*Create a Map with Layers and render to a Graphics2D object.*

```
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]
)
BufferedImage image = new BufferedImage(800, 300, BufferedImage.TYPE_INT_ARGB)
Graphics2D graphics = image.graphics
map.render(graphics)
graphics.dispose()
```

*Create a Map with Layers and display in a simple UI.*

```
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]
)
map.display()
```

# Map Properties

*Get Map properties*

```
Workspace workspace = new GeoPackage('src/main/resources/data.gpkg')
Layer countries = workspace.get("countries")
countries.style = new Fill("#ffffff") + new Stroke("#b2b2b2", 0.5)
Map map = new Map(
        width: 600,
        height: 600,
        backgroundColor: "#a5bfdd",
        layers: [countries],
        type: "png",
        proj: "EPSG:3857",
        bounds: new Bounds(-180,-85,180,85, "EPSG:4326").reproject("EPSG:3857"),
        fixAspectRatio: false
)
File file = new File("map.png")
map.render(file)
```

*Get width and height*

```
int width = map.width
int height = map.height
println "Width and Height = ${width} x ${height}"
```

```
Width and Height = 600 x 600
```

*Get the Bounds*

```
Bounds bounds = map.bounds
println "Bounds = ${bounds}"
```

```
Bounds = (-2.0037508342789244E7,-
1.9971868880408555E7,2.0037508342789244E7,1.9971868880408563E7,EPSG:3857)
```

*Get the Projection*

```
Projection projection = map.proj
println "Projeciton = ${projection}"
```

```
Projeciton = EPSG:3857
```

*Get the Layers*

```
List<Layer> layers = map.layers
println "Layers:"
layers.each { Layer layer ->
    println "   ${layer.name}"
}
```

```
Layers:
   countries
```

*Get the renderer type*

```
String type = map.type
println "Type = ${type}"
```

```
Type = png
```

*Get whether we are fixing the aspect ration or not.*

```
boolean shouldFixAspectRation = map.fixAspectRatio
println "Fix Aspect Ratio = ${shouldFixAspectRation}"
```

```
Fix Aspect Ratio = false
```

*Get the background color*

```
String backgroundColor = map.backgroundColor
println "Background Color = ${backgroundColor}"
```

```
Background Color = #a5bfdd
```

*Get the scale*

```
double scale = map.scaleDenominator
println "Scale = ${scale}"
```

```
Scale = 2.385417659855862E8
```

# Advanced Properties

*You can set the scale computation to be accurate (the default) or ogc compliant.*

```
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: 400,
        height: 300,
        layers: [ocean, countries],
        bounds: new Bounds(-162.070313,9.968851,-35.507813,58.995311, "EPSG:4326")
)

map.setScaleComputation("accurate")
File accurateFile = new File("map_accurate.png")
map.render(accurateFile)

map.setScaleComputation("ogc")
File ogcFile = new File("map_ogc.png")
map.render(ogcFile)
```

Accurate

OGC

*You can set whether to use advanced projection handling or not. By default this is set to true.*

```
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: 400,
        height: 300,
        layers: [ocean, countries],
        bounds: new Bounds(-162.070313,9.968851,-35.507813,58.995311, "EPSG:4326")
)

map.setAdvancedProjectionHandling(true)
File trueFile = new File("map_advancedproj_true.png")
map.render(trueFile)

map.setAdvancedProjectionHandling(false)
File falseFile = new File("map_advancedproj_false.png")
map.render(falseFile)
```

Yes

No

*You can set whether to use continuous map wrapping. The default is true.*

```
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: 200,
        layers: [ocean, countries]
)

map.setContinuousMapWrapping(true)
File trueFile = new File("map_continuouswrapping_true.png")
map.render(trueFile)

map.setContinuousMapWrapping(false)
File falseFile = new File("map_continuouswrapping_false.png")
map.render(falseFile)
```

Yes

No

# 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)
```

*Render a Map to an animated GIF to a byte array 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")
byte[] bytes = gif.renderAnimated(images, 500, true)
file.bytes = bytes
```

## 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 Polygon Layer using an Expression*

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

*Create a Label for a Polygon Layer using an Expression that concatenates properties and strings.*

```
Expression expression = Expression.fromCQL("Concatenate(z, '/', x, '/', y)")
Symbolizer symbolizer = new Stroke("black", 1.0) + new Label(expression)
```

*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
}
```

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

*Remove a Layer from a Workspace*

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

```
false
```

*Close the Workspace when you are done*

```
workspace.close()
```

# Creating an in Memory Workspace

*The empty Workspace constructor creates an in Memory Workspace. You can create a Layer by passing a name and a list of Fields. You can then remove the Layer by passing a reference to the Layer.*

```
Workspace workspace = new Workspace()

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

workspace.remove(layer)
println workspace.has(layer.name)
```

```
cities
false
```

# Add Layer's Features in Chunks

*When adding a large Layer to a Workspace, you can add Features in chunks.*

```
Workspace workspace = new Memory()
Layer layer = workspace.create("points", [
        new Field("geom", "Point", "EPSG:4326"),
        new Field("id", "Integer")
])
Bounds bounds = new Bounds(-180,-90, 180,90, "EPSG:4326")
Geometry.createRandomPoints(bounds.geometry, 500).geometries.eachWithIndex { Geometry
geom, int i ->
    layer.add([geom: geom, id: i])
}
println "Original Layer has ${layer.count} features."

Layer copyOfLayer = workspace.add(layer, "random points", 100)
println "Copied Layer has ${copyOfLayer.count} features."
```

```
Original Layer has 500 features.
Copied Layer has 500 features.
```

## 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
}
```

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

*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 = 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 = fetch size Type = java.lang.Integer Required? false
Parameter = Batch insert size 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)
```

*You can use the withWorkspace method to automatically handle closing the Workspace.*

```
Workspace.withWorkspace("dbtype=geopkg database=src/main/resources/data.gpkg") {
Workspace workspace ->
    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 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)
```

*You can use the withWorkspace method to automatically handle closing the Workspace.*

```
Workspace.withWorkspace([dbtype: 'geopkg', database: 'src/main/resources/data.gpkg'])
{ Workspace workspace ->
    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 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/cultural/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)
```

*Create a H2 Workspace with basic parameters*

```
H2 h2 = new H2(
        "database",    ①
        "localhost",   ②
        "5421",        ③
        "geo",         ④
        "user",        ⑤
        "password"     ⑥
)
```

① Database name

② Host name

③ Port

④ User name

⑤ Password

*Create a H2 Workspace with named parameters. Only the database name is required.*

```
H2 h2 = new H2("database",
        "host": "localhost",
        "port": "5412",
        "schema": "geo",
        "user": "user",
        "password": "secret"
)
```

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

## 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)
```

## Creating PostGIS Workspaces

*Create a PostGIS Workspace with basic parameters*

```
PostGIS postgis = new PostGIS(
        "database",    ①
        "localhost",   ②
        "5432",        ③
        "public",      ④
        "user",        ⑤
        "password"     ⑥
)
```

① Database name

② Host name

③ Port

④ Schema

⑤ User name

⑥ Password

*Create a PostGIS Workspace with advanced parameters*

```
PostGIS postgis = new PostGIS(
        "database",                  ①
        "localhost",                 ②
        "5432",                      ③
        "public",                    ④
        "user",                      ⑤
        "password",                  ⑥
        true,                        ⑦
        true,                        ⑧
        "OWNER geo TABLESPACE points" ⑨
)
```

① Database name

② Host name

③ Port

④ Schema

⑤ User name

⑥ Password

⑦ Estimated Extent

⑧ Create Database

⑨ Create Database Params

*Create a PostGIS Workspace with named parameters. Only the database name is required.*

```
PostGIS postgis = new PostGIS("database",
        "host": "localhost",
        "port": "5432",
        "schema": "public",
        "user": "user",
        "password": "secret",
        "estimatedExtent": false,
        "createDatabase": false,
        "createDatabaseParams": "OWNER geo TABLESPACE points"
)
```

*Delete a PostGIS database.*

```
PostGIS.deleteDatabase(
        "database",    ①
        "localhost",   ②
        "5432",        ③
        "user",        ④
        "password"     ⑤
)
```

① Database name

② Host name

③ Port

④ User name

⑤ Password

*Delete a PostGIS database with named parameters. Only the database name is required.*

```
PostGIS.deleteDatabase("database",
        "host": "localhost",
        "port": "5432",
        "user": "user",
        "password": "secret"
)
```

## Creating MySQL Workspaces

*Create a MySQL Workspace with basic parameters*

```
MySQL mysql = new MySQL(
        "database",    ①
        "localhost",   ②
        "3306",        ③
        "user",        ④
        "password"     ⑤
)
```

① Database name

② Host name

③ Port

④ User name

⑤ Password

*Create a MySQL Workspace with named parameters.  Only the database name is required.*

```
MySQL mysql = new MySQL("database",
        "host": "localhost",
        "port": "3306",
        "user": "user",
        "password": "secret"
)
```

## Creating SpatiaLite Workspaces

*Create a SpatiaLite Workspace with file and directory name*

```
SpatiaLite spatialite = new SpatiaLite(
        "db.sqlite", ①
        "databases"  ②
)
```

① File name

② Directory name

*Create a SpatiaLite Workspace with file and a File directory*

```
File directory = new File("databases")
SpatiaLite spatialite = new SpatiaLite(
        "db.sqlite", ①
        directory    ②
)
```

① File name

② File Directory

### Creating WFS Workspaces

*Create a WFS Workspace with a URL*

```
WFS wfs = new WFS
("http://localhost:8080/geoserver/ows?service=wfs&version=1.1.0&request=GetCapabilitie
s")
```

### Creating OGR Workspaces

*Determine if OGR is available.*

```
boolean isAvailable = OGR.isAvailable()
```

*Get OGR Drivers.*

```
Set<String> drivers = OGR.drivers
```

*Get a Shapefile Workspace from OGR.*

```
File file = new File("states.shp")
OGR ogr = new OGR("ESRI Shapefile", file.absolutePath)
```

*Get a SQLite Workspace from OGR*

```
File file = new File("states.sqlite")
OGR ogr = new OGR("SQLite", file.absolutePath)
```

*Get a GeoJSON Workspace from OGR*

```
File file = new File("states.json")
OGR ogr = new OGR("GeoJSON", file.absolutePath)
```

## Database Workspace

### SQL

*Run SQL queries directly against Databse Workspace (PostGIS, MySQL, H2)*

```
Database workspace = new H2(new File("src/main/resources/h2/data.db"))
Sql sql = workspace.sql
```

*Count the number of results*

```
int numberOfPlaces = sql.firstRow("SELECT COUNT(*) as count FROM \"places\"").get
("count") as int
println "# of Places = ${numberOfPlaces}"
```

```
# of Places = 326
```

*Calculate statistics*

```
GroovyRowResult result = sql.firstRow("SELECT MIN(ELEVATION) as min_elev,
MAX(ELEVATION) as max_elev, AVG(ELEVATION) as avg_elev FROM \"places\"")
println "Mininum Elevation = ${result.get('min_elev')}"
println "Maximum Elevation = ${result.get('max_elev')}"
println "Average Elevation = ${result.get('avg_elev')}"
```

```
Mininum Elevation = 0.0
Maximum Elevation = 2320.0
Average Elevation = 30.085889570552148
```

*Select rows*

```
List<String> names = []
sql.eachRow "SELECT TOP 10 \"NAME\" FROM \"places\" ORDER BY \"NAME\" DESC ", {
    names.add(it["NAME"])
}
names.each { String name ->
    println name
}
```

```
 Zürich
Zibo
Zhengzhou
Zagreb
Yerevan
Yaounde
Yamoussoukro
Xian
Wuhan
Windhoek
```

*Execute spatial sql*

```
Workspace memory = new Memory()
Layer layer = memory.create("places_polys", [new Field("buffer", "Polygon"), new
Field("name", "String")])
sql.eachRow "SELECT ST_Buffer(\"the_geom\", 10) as buffer, \"NAME\" as name FROM
\"places\"", {row ->
    Geometry poly = Geometry.fromWKB(row.buffer as byte[])
    layer.add([buffer: poly, name: row.NAME])
}
```

## View

*Create a new Layer from a SQL View*

```
Database workspace = new H2(new File("src/main/resources/h2/data.db"))
Layer layer = workspace.createView(
        "megacities",                                          ①
        "SELECT * FROM \"places\" WHERE \"MEGACITY\" = '%mega%'", ②
        new Field("the_geom","Point","EPSG:4326"),             ③
        params: [['mega', '1']]                                ④
)
boolean hasLayer1 = workspace.has("megacities")
println "Does layer exist? ${hasLayer1}"
```

① The layer name

② The SQL Statement

③ The Geometry Field

④ Query Parameters

```
Does layer exist? true
```

*Remove the new Layer created from a SQL View*

```
workspace.deleteView("megacities")
boolean hasLayer2 = workspace.has("megacities")
println "Does layer exist? ${hasLayer2}"
```

```
Does layer exist? false
```

## Index

*Create an Index*

```
Database workspace = new H2(new File("src/main/resources/h2/data.db"))
workspace.createIndex("places","name_idx","NAME",true)
workspace.createIndex("places","megacity_idx","MEGACITY",false)
workspace.createIndex("places","a3_idx", ["SOV_A3", "ADM0_A3"],false)
```

*Get an Index*

```
List<Map> indexes = workspace.getIndexes("places")
indexes.each { Map index ->
    println "Index name = ${index.name}, unique = ${index.unique}, attributes =
${index.attributes}"
}
```

```
Index name = PRIMARY_KEY_C, unique = true, attributes = [fid]
Index name = name_idx, unique = true, attributes = [NAME]
Index name = a3_idx, unique = false, attributes = [SOV_A3, ADM0_A3]
Index name = megacity_idx, unique = false, attributes = [MEGACITY]
```

*Remove an Index*

```
workspace.deleteIndex("places", "name_idx")
workspace.deleteIndex("places", "megacity_idx")
workspace.deleteIndex("places", "a3_idx")
```

# 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

## Each Feature

*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 Features

*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

*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
```

## First

*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
```

## Filter

*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'")
```

## Cursor

*Iterate over a Layer's Features with a Cursor*

```
Workspace workspace = new GeoPackage("src/main/resources/data.gpkg")
Layer layer = workspace.get("states")
Cursor cursor = layer.cursor
cursor.each { 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 with a Cursor*

```
Workspace workspace = new GeoPackage("src/main/resources/data.gpkg")
Layer layer = workspace.get("states")
Cursor cursor = layer.getCursor(filter: "NAME_1 LIKE 'M%'")
while(cursor.hasNext()) {
    Feature feature = cursor.next()
    println feature["NAME_1"]
}
```

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

*Iterate over a Layer's Features with parameters with a Cursor*

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

# 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 |

*Add Features to a Layer using a Writer. A Writer can add Features more effeciently because is commits batches of Features in Transactions .*

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

Bounds bounds = new Bounds(-180,-90,180,90, "EPSG:4326")
MultiPoint points = Geometry.createRandomPoints(bounds.geometry, 100)

geoscript.layer.Writer writer = layer.writer
try {
    points.geometries.eachWithIndex { Point point, int index ->
        Feature feature = writer.newFeature
        feature['id'] = index
        feature['geom'] = point
        writer.add(feature)
    }
} finally {
    writer.close()
}
```

*Add Features to a Layer using a Writer inside of a Closure.*

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

Bounds bounds = new Bounds(-180,-90,180,90, "EPSG:4326")
MultiPoint points = Geometry.createRandomPoints(bounds.geometry, 100)

layer.withWriter { geoscript.layer.Writer writer ->
    points.geometries.eachWithIndex { Point point, int index ->
        Feature feature = writer.newFeature
        feature['id'] = index
        feature['geom'] = point
        writer.add(feature)
    }
}
```

# Shapefiles

Shapefiles are a very commonly used format for storing spatial data. So, instead of creating a Directory Workspace and getting the Layer from the Workspace, you can use the Shapefile class.

## Read

*Read existing Shapefiles*

```
Shapefile countries = new Shapefile("src/main/resources/shapefiles/countries.shp")
println "# Features in Countries = ${countries.count}"

Shapefile ocean = new Shapefile(new File("src/main/resources/shapefiles/ocean.shp"))
println "# Features in Ocean = ${ocean.count}"
```

```
# Features in Countries = 177

# Features in Ocean = 2
```

## Create

*Create a new Shapefile*

```
Directory workspace = new Directory("target")
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)
```

| id | name | state |
|---|---|---|
| 1 | Seattle | WA |
| 2 | Tacoma | WA |
| 3 | Fargo | ND |

| id | name | state |
|---|---|---|
| 4 | Bismarck | ND |
| 5 | Mandan | ND |

# Property

GeoScript can store spatial data in a simple plain text format. With the Property class you can access a single property file directly.

## Read

*Read existing Property files*

```
geoscript.layer.Property circles = new geoscript.layer.Property
("src/main/resources/property/circles.properties")
println "# Features in circles = ${circles.count}"

geoscript.layer.Property places = new geoscript.layer.Property(new File
("src/main/resources/property/places.properties"))
println "# Features in places = ${places.count}"
```

```
# Features in circles = 10

# Features in places = 10
```

## Create

*Create a new Property*

```
geoscript.workspace.Property workspace = new geoscript.workspace.Property("target")
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)
```

| id | name | state |
|---|---|---|
| 1 | Seattle | WA |
| 2 | Tacoma | WA |
| 3 | Fargo | ND |

| id | name | state |
|---|---|---|
| 4 | Bismarck | ND |
| 5 | Mandan | ND |

# 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 Layers

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-2114f051_168da228c02_-6903"
        },
        {
            "type": "Feature",
            "geometry": {
                "type": "Point",
                "coordinates": [
                    -122.4842,
                    47.2619
                ]
            },
            "properties": {
                "id": 2,
                "name": "Tacoma"
            },
            "id": "fid-2114f051_168da228c02_-6901"
        }
    ]
}
```

*Write a Layer to a GeoJSON String*

```
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"
])

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

```
{
    "type": "FeatureCollection",
    "features": [
        {
            "type": "Feature",
            "geometry": {
                "type": "Point",
                "coordinates": [
                    -122.3204,
                    47.6024
                ]
            },
            "properties": {
                "id": 1,
                "name": "Seattle"
            },
            "id": "fid-2114f051_168da228c02_-7e2d"
        },
        {
            "type": "Feature",
            "geometry": {
                "type": "Point",
                "coordinates": [
                    -122.4842,
                    47.2619
                ]
            },
            "properties": {
                "id": 2,
                "name": "Tacoma"
            },
            "id": "fid-2114f051_168da228c02_-7e2b"
        }
    ]
}
```

*Read a Layer from a GeoJSON String*

```
        String geoJson = """
{
    "type": "FeatureCollection",
    "features": [
        {
            "type": "Feature",
            "geometry": {
                "type": "Point",
                "coordinates": [
                    -122.3204,
                    47.6024
                ]
            },
            "properties": {
                "id": 1,
                "name": "Seattle"
            },
            "id": "1"
        },
        {
            "type": "Feature",
            "geometry": {
                "type": "Point",
                "coordinates": [
                    -122.681944,
                    45.52
                ]
            },
            "properties": {
                "id": 2,
                "name": "Portland"
            },
            "id": "2"
        }
    ]
}
"""
        GeoJSONReader reader = new GeoJSONReader()
        Layer layer = reader.read(geoJson)
```

## 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
```

*Write a Layer to a GeoBuf String*

```
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(-122.681944, 45.52),
        id: 2,
        name: "Portland"
])

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

```
0a0269640a046e616d6510021806223f0a1d0a0c08001a089fd8d374c0ebb22d6a0218016a090a07536561
74746c650a1e0a0c08001a08afe9ff7480d2b42b6a0218026a0a0a08506f72746c616e64
```

*Read a Layer from a GeoBuf String*

```
String geobuf =
"0a0269640a046e616d6510021806223f0a1d0a0c08001a089fd8d374c0ebb22d6a0218016a090a0753656
174746c650a1e0a0c08001a08afe9ff7480d2b42b6a0218026a0a0a08506f72746c616e64"
GeobufReader reader = new GeobufReader()
Layer layer = reader.read(geobuf)
```

## 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-
2114f051_168da228c02_-7398">
      <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-
2114f051_168da228c02_-7396">
      <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>
```

*Write a Layer to a GML String*

```
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"
])

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

```
<wfs:FeatureCollection xmlns:gsf="http://geoscript.org/feature"
xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:wfs="http://www.opengis.net/wfs"
xmlns:gml="http://www.opengis.net/gml" xmlns:ogc="http://www.opengis.net/ogc">
<gml:boundedBy>
<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>
<gsf:cities fid="fid-2114f051_168da228c02_-73b2">
<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>
<gsf:cities fid="fid-2114f051_168da228c02_-73b0">
<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>
```

*Read a Layer from a GML String*

```
        String gml = """
<wfs:FeatureCollection xmlns:gsf="http://geoscript.org/feature"
xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:wfs="http://www.opengis.net/wfs"
xmlns:gml="http://www.opengis.net/gml" xmlns:ogc="http://www.opengis.net/ogc">
    <gml:boundedBy>
        <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>
        <gsf:cities fid="fid-a7cd555_1634fc34503_-7fff">
            <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>
        <gsf:cities fid="fid-a7cd555_1634fc34503_-7ffd">
            <gml:name>Portland</gml:name>
            <gsf:geom>
                <gml:Point>
                    <gml:coord>
                        <gml:X>-122.681944</gml:X>
                        <gml:Y>45.52</gml:Y>
                    </gml:coord>
                </gml:Point>
            </gsf:geom>
            <gsf:id>2</gsf:id>
        </gsf:cities>
    </gml:featureMember>
</wfs:FeatureCollection>
"""
        GmlReader reader = new GmlReader()
        Layer layer = reader.read(gml)
```

## 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-2114f051_168da228c02_-6a0e
      </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-2114f051_168da228c02_-6a0c
      </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>
```

*Write a Layer to a KML String*

```
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"
])

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

```
<kml:kml xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:kml="http://earth.google.com/kml/2.1">
<kml:Document>
<kml:Placemark id="fid-2114f051_168da228c02_-73aa">
<kml:name>Seattle</kml:name>
<kml:Point>
<kml:coordinates>-122.3204,47.6024</kml:coordinates>
</kml:Point>
</kml:Placemark>
<kml:Placemark id="fid-2114f051_168da228c02_-73a8">
<kml:name>Tacoma</kml:name>
<kml:Point>
<kml:coordinates>-122.48416,47.2619</kml:coordinates>
</kml:Point>
</kml:Placemark>
</kml:Document>
</kml:kml>
```

*Read a Layer from a KML String*

```
        String kml = """
<kml:kml xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:kml="http://earth.google.com/kml/2.1">
    <kml:Document>
        <kml:Placemark id="fid-61215c1b_1634ca279f5_-7fff">
            <kml:name>Seattle</kml:name>
            <kml:Point>
                <kml:coordinates>-122.3204,47.6024</kml:coordinates>
            </kml:Point>
        </kml:Placemark>
        <kml:Placemark id="fid-61215c1b_1634ca279f5_-7ffd">
            <kml:name>Portland</kml:name>
            <kml:Point>
                <kml:coordinates>-122.681944,45.52</kml:coordinates>
            </kml:Point>
        </kml:Placemark>
    </kml:Document>
</kml:kml>
"""
        KmlReader reader = new KmlReader()
        Layer layer = reader.read(kml)
```

## CSV

*Write a Layer to a CSV String*

```
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"
])

CsvWriter writer = new CsvWriter()
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"
```

*Read a Layer from a CSV String*

```
        String csv = """"geom:Point:EPSG:4326","id:Integer","name:String"
"POINT (-122.3204 47.6024)","1","Seattle"
"POINT (-122.681944 45.52)","2","Portland"
"""
        CsvReader reader = new CsvReader()
        Layer layer = reader.read(csv)
```

## GeoRSS

*Write a Layer to a GeoRSS String*

```
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"
])

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

```
<?xml version="1.0" encoding="UTF-8"?><feed
xmlns:georss="http://www.georss.org/georss" xmlns="http://www.w3.org/2005/Atom">
<title>cities</title>
<subtitle>cities geom: Point(EPSG:4326), id: Integer, name: String</subtitle>
<link>http://geoscript.org/feature</link>
<entry>
<title>fid-2114f051_168da228c02_-7a36</title>
<summary>[geom:POINT (-122.3204 47.6024), id:1, name:Seattle]</summary>
<updated>Mon Feb 11 01:18:58 UTC 2019</updated>
<georss:point>47.6024 -122.3204</georss:point>
</entry>
<entry>
<title>fid-2114f051_168da228c02_-7a34</title>
<summary>[geom:POINT (-122.48416 47.2619), id:2, name:Tacoma]</summary>
<updated>Mon Feb 11 01:18:58 UTC 2019</updated>
<georss:point>47.2619 -122.48416</georss:point>
</entry>
</feed>
```

*Read a Layer from a GeoRSS String*

```
        String georss = """<?xml version="1.0" encoding="UTF-8"?>
<feed xmlns:georss="http://www.georss.org/georss" xmlns="http://www.w3.org/2005/Atom">
<title>cities</title>
<subtitle>cities geom: Point(EPSG:4326), id: Integer, name: String</subtitle>
<link>http://geoscript.org/feature</link>
<entry>
<title>Seattle</title>
<summary>[geom:POINT (-122.3204 47.6024), id:1, name:Seattle]</summary>
<updated>Fri May 11 15:23:05 PDT 2018</updated>
<georss:point>47.6024 -122.3204</georss:point>
</entry>
<entry>
<title>Portland</title>
<summary>[geom:POINT (-122.681944 45.52), id:2, name:Portland]</summary>
<updated>Fri May 11 15:23:05 PDT 2018</updated>
<georss:point>45.52 -122.681944</georss:point>
</entry>
</feed>
"""
        GeoRSSReader reader = new GeoRSSReader()
        Layer layer = reader.read(georss)
```

## GPX

*Write a Layer to a GPX String*

```
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"
])

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

```
<?xml version="1.0" encoding="UTF-8"?><gpx xmlns="http://www.topografix.com/GPX/1/1"
version="1.1" creator="geoscript">
<wpt lat="47.6024" lon="-122.3204">
<name>fid-2114f051_168da228c02_-73ae</name>
</wpt>
<wpt lat="47.2619" lon="-122.48416">
<name>fid-2114f051_168da228c02_-73ac</name>
</wpt>
</gpx>
```

*Read a Layer from a GPX String*

```
        String gpx = """<?xml version="1.0" encoding="UTF-8"?>
<gpx xmlns="http://www.topografix.com/GPX/1/1" version="1.1" creator="geoscript">
<wpt lat="47.6024" lon="-122.3204">
<name>Seattle</name>
</wpt>
<wpt lat="45.52" lon="-122.681944">
<name>Portland</name>
</wpt>
</gpx>
"""
        GpxReader reader = new GpxReader()
        Layer layer = reader.read(gpx)
```

# 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
```

# GeoTIFF

## Read

Read a GeoTIFF Raster from a File

```
File file = new File("src/main/resources/earth.tif")
GeoTIFF geotiff = new GeoTIFF(file)
Raster raster = geotiff.read("earth")
```

### Write

Write a GeoTIFF Raster to an ArcGrid Raster

```
File file = new File("src/main/resources/earth.tif")
GeoTIFF geotiff = new GeoTIFF(file)
Raster raster = geotiff.read("earth")

File arcGridFile = new File("target/earth.asc")
ArcGrid arcGrid = new ArcGrid(arcGridFile)
arcGrid.write(raster)
Raster arcGridRaster = arcGrid.read("earth")
```

## WorldImage

### Read

Read a WorldImage Raster from a File

```
File file = new File("src/main/resources/earth.png")
WorldImage worldImage = new WorldImage(file)
Raster raster = worldImage.read("earth")
```

# 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 = 255.0
  No Data = [0.0]
  Is No Data = false
  Unit = null
  Scale = 1.0
  Offset = 0.0
  Type = byte

GREEN_BAND
  Min = 0.0
  Max = 255.0
  No Data = [0.0]
  Is No Data = false
  Unit = null
  Scale = 1.0
  Offset = 0.0
  Type = byte

BLUE_BAND
  Min = 0.0
  Max = 255.0
  No Data = [0.0]
  Is No Data = false
  Unit = null
  Scale = 1.0
  Offset = 0.0
  Type = byte
```

Get the minimum and maximum values from a Raster for each band

```
Map extrema = raster.extrema
println "Min values: ${extrema.min} Max values: ${extrema.max}"
```

```
Min value: [56.0, 84.0, 91.0] Max value: [255.0, 255.0, 255.0]
```

## 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
```

Set values on a Raster

```
File file = new File("src/main/resources/earth.tif")
GeoTIFF geotiff = new GeoTIFF(file)
Raster raster = geotiff.read("earth")

File arcGridFile = new File("target/earth.asc")
ArcGrid arcGrid = new ArcGrid(arcGridFile)
arcGrid.write(raster)
Raster arcGridRaster = arcGrid.read("earth")

arcGridRaster.eachCell {double value, double x, double y ->
    double newValue = value + 100
    arcGridRaster.setValue([x as int, y as int], newValue)
}

File arcGridAddFile = new File("target/earth_100.asc")
ArcGrid arcGridAdd = new ArcGrid(arcGridAddFile)
arcGridAdd.write(arcGridRaster)
Raster arcGridRasterAdd = arcGridAdd.read("earth_100")

List pixels = [
    [92, 298],
    [393.0, 343.0],
    [795.0, 399.0]
]
pixels.each { List pixel ->
    println "Original: ${raster.getValue(pixel)} New:
${arcGridRasterAdd.getValue(pixel)}"
}
```

```
Original: 97.0 New: 197.0
Original: 96.0 New: 196.0
Original: 237.0 New: 337.0
```

# 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)
```

## Contours

Create vector contours from a Raster

```
File file = new File("src/main/resources/pc.tif")
Format format = Format.getFormat(file)
Raster raster = format.read("pc")
int band = 0
int interval = 300
boolean simplify = true
boolean smooth = true
Layer contours = raster.contours(band, interval, simplify, smooth)
```

## Stylize

Stylize a Raster by baking in a style to create a new Raster

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

## Reclassify

Reclassify a Raster

```
File file = new File("src/main/resources/pc.tif")
Format format = Format.getFormat(file)
Raster raster = format.read("pc")
Raster reclassifiedRaster = raster.reclassify([
        [min:0,    max:0,    value: 1],
        [min:0,    max:50,   value: 2],
        [min:50,   max:200,  value: 3],
        [min:200,  max:1000, value: 4],
        [min:1000, max:1500, value: 5],
        [min:1500, max:4000, value: 6]
])
```

## Scale

Scale a Raster

```
File file = new File("src/main/resources/pc.tif")
Format format = Format.getFormat(file)
Raster raster = format.read("pc")
println "Original Raster Size = ${raster.size[0]}x${raster.size[1]}"

Raster scaledRaster = raster.scale(0.5, 0.5)
println "Scaled Raster Size = ${scaledRaster.size[0]}x${scaledRaster.size[1]}"
```

```
Original Raster Size = 800x400
Scaled Raster Size = 400x200
```

## Invert

Invert the values of a Raster

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

## Exponent

Calculate the exponent of the values of a Raster

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

## Log

Calculate the log of the values of a Raster

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

## Vectorize

Create a Polygon Layer from a Raster

```
File file = new File("src/main/resources/pc.tif")
Format format = Format.getFormat(file)
Raster raster = format.read("pc")
Raster reclassifiedRaster = raster.reclassify([
        [min:0,    max:0,    value: 1],
        [min:0,    max:50,   value: 2],
        [min:50,   max:200,  value: 3],
        [min:200,  max:1000, value: 4],
        [min:1000, max:1500, value: 5],
        [min:1500, max:4000, value: 6]
])
Layer layer = reclassifiedRaster.polygonLayer
```

Create a Point Layer from a Raster

```
File file = new File("src/main/resources/pc.tif")
Format format = Format.getFormat(file)
Raster raster = format.read("pc").crop(new Bounds(-121.878548,46.808402,-121.636505
,46.896097, "EPSG:4326"))
Layer layer = raster.pointLayer
```

## Raster Algebra

### Add

Add a constant value to a Raster

```
File file = new File("src/main/resources/pc.tif")
Format format = Format.getFormat(file)
Raster raster = format.read("pc")
double elevation1 = raster.getValue(new Point(-121.799927,46.867703))
println elevation1

Raster higherRaster = raster.add(100.00)
double elevation2 = higherRaster.getValue(new Point(-121.799927,46.867703))
println elevation2
```

```
3069.0
3169.0
```

### Subtract

Subtract a constant value from a Raster

```
File file = new File("src/main/resources/pc.tif")
Format format = Format.getFormat(file)
Raster raster = format.read("pc")
double elevation1 = raster.getValue(new Point(-121.799927,46.867703))
println elevation1

Raster lowerRaster = raster.minus(50.00)
double elevation2 = lowerRaster.getValue(new Point(-121.799927,46.867703))
println elevation2
```

```
3069.0
3019.0
```

Subtract the Raster from a constant value

```
File file = new File("src/main/resources/pc.tif")
Format format = Format.getFormat(file)
Raster raster = format.read("pc")
double elevation1 = raster.getValue(new Point(-121.799927,46.867703))
println elevation1

Raster lowerRaster = raster.minusFrom(2000.0)
double elevation2 = lowerRaster.getValue(new Point(-121.799927,46.867703))
println elevation2
```

```
3069.0
-1069.0
```

### Multiply

Multiply a constant value against a Raster

```
File file = new File("src/main/resources/pc.tif")
Format format = Format.getFormat(file)
Raster raster = format.read("pc")
double elevation1 = raster.getValue(new Point(-121.799927,46.867703))
println elevation1

Raster higherRaster = raster.multiply(2.0)
double elevation2 = higherRaster.getValue(new Point(-121.799927,46.867703))
println elevation2
```

```
3069.0
6138.0
```

### Divide

Divide a constant value against a Raster

```
File file = new File("src/main/resources/pc.tif")
Format format = Format.getFormat(file)
Raster raster = format.read("pc")
double elevation1 = raster.getValue(new Point(-121.799927,46.867703))
println elevation1

Raster lowerRaster = raster.divide(2.0)
double elevation2 = lowerRaster.getValue(new Point(-121.799927,46.867703))
println elevation2
```

```
3069.0
1534.5
```

# 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 Bounds from a Pyramid

Get the Bounds for a Tile.

```
Pyramid pyramid = Pyramid.createGlobalMercatorPyramid()
Tile tile = new Tile(2, 1, 1)
Bounds bounds = pyramid.bounds(tile)
println "The bounds of ${tile} is ${bounds}"
```

```
The bounds of Tile(x:1, y:1, z:2) is (-1.0018197573940657E7,-
1.0018735602568535E7,0.0,-3.725290298461914E-9,EPSG:3857)
```

Get the Bounds for an area around a Point at a zoom level.

```
Pyramid pyramid = Pyramid.createGlobalMercatorPyramid()
Point point = Projection.transform(new Point(22.1539306640625, 37.67077737288316),
"EPSG:4326", "EPSG:3857")
int zoomLevel = 8
int width = 400
int height = 400
Bounds bounds = pyramid.bounds(point, zoomLevel, width, height)
println "The bounds around ${point} is ${bounds}"
```

```
The bounds around POINT (2466164.2805929263 4533021.525424092) is
(2343967.4055929263,4410824.650424092,2588361.1555929263,4655218.400424092,EPSG:3857)
```

## 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
```

Get a Grid from a Pyramid by a Bounds and Resolution.

```
Pyramid pyramid = Pyramid.createGlobalMercatorPyramid()
Bounds bounds = new Bounds(-123.09, 46.66, -121.13, 47.48, "EPSG:4326").reproject
("EPSG:3857")
Grid grid = pyramid.grid(bounds, bounds.width / 400.0, bounds.height / 200.0)
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: 8
Width / # Columns: 256
Height / # Rows: 256
Size / # Tiles: 65536
X Resolution: 610.984375
Y Resolution: 610.984375
```

Get a Grid from a Pyramid by a Bounds and Size.

```
Pyramid pyramid = Pyramid.createGlobalMercatorPyramid()
Bounds bounds = new Bounds(-123.09, 46.66, -121.13, 47.48, "EPSG:4326").reproject
("EPSG:3857")
Grid grid = pyramid.grid(bounds, 400, 200)
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: 8
Width / # Columns: 256
Height / # Rows: 256
Size / # Tiles: 65536
X Resolution: 610.984375
Y Resolution: 610.984375
```

## Get Tile Coordinates

Get the tile coordinates from a Pyramid by Bounds and zoom level

```
Pyramid pyramid = Pyramid.createGlobalMercatorPyramid()
Bounds bounds = new Bounds(-124.73142200000001, 24.955967, -66.969849, 49.371735,
"EPSG:4326").reproject("EPSG:3857")
long zoomLevel = 4
Map<String, Integer> coords = pyramid.getTileCoordinates(bounds, zoomLevel)
println "Min X = ${coords.minX}"
println "Min Y = ${coords.minY}"
println "Max X = ${coords.maxX}"
println "Max Y = ${coords.maxY}"
```

```
Min X = 2
Min Y = 9
Max X = 5
Max Y = 10
```

Get the tile coordinates from a Pyramid by Bounds and Grid

```
Pyramid pyramid = Pyramid.createGlobalMercatorPyramid()
Bounds bounds = new Bounds(20.798492,36.402494,22.765045,37.223768, "EPSG:4326"
).reproject("EPSG:3857")
Grid grid = pyramid.grid(10)
Map<String, Integer> coords = pyramid.getTileCoordinates(bounds, grid)
println "Min X = ${coords.minX}"
println "Min Y = ${coords.minY}"
println "Max X = ${coords.maxX}"
println "Max Y = ${coords.maxY}"
```

```
Min X = 571
Min Y = 623
Max X = 576
Max Y = 626
```

## 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
```

```
        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,
-2.0037471205137067E7,2.0036395147881314E7,2.003747120513706E7,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
```

**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

### MBTiles

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

**GeoPackage**

```
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)
```

Generate Image Tiles to a TMS directory

**TMS**

```
File directory = new File("target/tiles")
directory.mkdir()
TMS tms = new TMS("world", "png", directory, Pyramid.createGlobalMercatorPyramid())

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(tms, [ocean, countries])
TileGenerator generator = new TileGenerator()
generator.generate(tms, 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,-
2.0037471205137067E7,2.0036395147881314E7,2.003747120513706E7,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)
```

## Get, put, and delete a Tile from a TileLayer

Get a Tile from a TileLayer.

```
MBTiles layer = new MBTiles(file)
ImageTile tile = layer.get(0,0,0)
```

Add a Tile to a TileLayer.

```
File newTileFile = new File("src/main/resources/yellowtile.png")
ImageTile newTile = new ImageTile(0,0,0, newTileFile.bytes)
layer.put(newTile)
newTile = layer.get(0,0,0)
```

Remove a Tile from a TileLayer.

```
layer.delete(newTile)
newTile = layer.get(0,0,0)
println "Image = ${newTile.image}"
```

```
Image = null
```

Close a TileLayer

```
layer.close()
```

## Delete Tiles from a TileLayer

```
MBTiles layer = new MBTiles(file)
layer.tiles(1).each { Tile tile ->
    println "${tile} = ${tile.image == null}"
}
```

```
Tile(x:0, y:0, z:1) = false
Tile(x:1, y:0, z:1) = false
Tile(x:0, y:1, z:1) = false
Tile(x:1, y:1, z:1) = false
```

```
layer.delete(layer.tiles(1))
layer.tiles(1).each { Tile tile ->
    println "${tile} = ${tile.image == null}"
}
```

```
Tile(x:0, y:0, z:1) = true
Tile(x:1, y:0, z:1) = true
Tile(x:0, y:1, z:1) = true
Tile(x:1, y:1, z:1) = true
```

### Tiles

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

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

long zoomLevel = 1
TileCursor tileCursor = mbtiles.tiles(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 from a TileLayer with Tiles from 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 = mbtiles.tiles(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 from a TileLayer for a zoom level and 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 = mbtiles.tiles(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 from a TileLayer for a zoom level and given x and y resolutions.

```
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 = mbtiles.tiles(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)
```

Get a TileCursor from a TileLayer for a Bounds and a given canvas width and height.

```
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 width = 400
int height = 400
TileCursor tileCursor = mbtiles.tiles(bounds, width, height)

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: 7
# of tiles: 6
Bounds: (-1.1583540944868885E7,5635538.7764447965,-
1.0644334922311949E7,6261709.751605329,EPSG:3857)
Width / # Columns: 3
Height / # Rows: 2
MinX: 27, MinY: 82, MaxX: 29, MaxY: 83

Tiles:
Tile(x:27, y:82, z:7)
Tile(x:28, y:82, z:7)
Tile(x:29, y:82, z:7)
Tile(x:27, y:83, z:7)
Tile(x:28, y:83, z:7)
Tile(x:29, y:83, z:7)
```

Get a TileCursor from a TileLayer around a Point at a given zoom level for a given canvas width and height.

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

Point point = Projection.transform(new Point(-102.875977, 45.433154), "EPSG:4326",
"EPSG:3857")
int zoomLevel = 12
int width = 400
int height = 400
TileCursor tileCursor = mbtiles.tiles(point, zoomLevel, width, height)

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: 12
# of tiles: 9
Bounds: (-1.1466140192049267E7,5674674.46239233,-
1.1436790003844364E7,5704026.226852979,EPSG:3857)
Width / # Columns: 3
Height / # Rows: 3
MinX: 876, MinY: 2628, MaxX: 878, MaxY: 2630

Tiles:
Tile(x:876, y:2628, z:12)
Tile(x:877, y:2628, z:12)
Tile(x:878, y:2628, z:12)
Tile(x:876, y:2629, z:12)
Tile(x:877, y:2629, z:12)
Tile(x:878, y:2629, z:12)
Tile(x:876, y:2630, z:12)
Tile(x:877, y:2630, z:12)
Tile(x:878, y:2630, z:12)
```

Get the tile coordinates from a TileLayer by Bounds and Grid

```
File file = new File("src/main/resources/tiles.mbtiles")
MBTiles mbtiles = new MBTiles(file)
Bounds bounds = new Bounds(20.798492,36.402494,22.765045,37.223768, "EPSG:4326"
).reproject("EPSG:3857")
Grid grid = mbtiles.pyramid.grid(10)
Map<String, Integer> coords = mbtiles.getTileCoordinates(bounds, grid)
println "Min X = ${coords.minX}"
println "Min Y = ${coords.minY}"
println "Max X = ${coords.maxX}"
println "Max Y = ${coords.maxY}"
```

```
Min X = 571
Min Y = 623
Max X = 576
Max Y = 626
```

Get a Layer from a TileLayer representing the outline of the Tiles in the TileCursor.

```
File file = new File("src/main/resources/tiles.mbtiles")
MBTiles mbtiles = new MBTiles(file)
Layer layer = mbtiles.getLayer(mbtiles.tiles(1))
```

| id | z | x | y |
|---|---|---|---|
| 0 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 |

| id | z | x | y |
|---|---|---|---|
| 2 | 1 | 0 | 1 |
| 3 | 1 | 1 | 1 |

# TileCursor

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

Get a TileCursor with all of the Tiles from 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 from 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 from 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 from 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)
```

Get a TileCursor with Tiles from a TileLayer within a Bounds for a given canvas width and height.

```
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 width = 400
int height = 400
TileCursor tileCursor = new TileCursor(mbtiles, bounds, width, height)

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: 7
# of tiles: 6
Bounds: (-1.1583540944868885E7,5635538.7764447965,-
1.0644334922311949E7,6261709.751605329,EPSG:3857)
Width / # Columns: 3
Height / # Rows: 2
MinX: 27, MinY: 82, MaxX: 29, MaxY: 83

Tiles:
Tile(x:27, y:82, z:7)
Tile(x:28, y:82, z:7)
Tile(x:29, y:82, z:7)
Tile(x:27, y:83, z:7)
Tile(x:28, y:83, z:7)
Tile(x:29, y:83, z:7)
```

Get a TileCursor with Tiles from a TileLayer around a Point at a given zoom level for a given canvas width and height.

```
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 width = 400
int height = 400
TileCursor tileCursor = new TileCursor(mbtiles, bounds, width, height)

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: 7
# of tiles: 6
Bounds: (-1.1583540944868885E7,5635538.7764447965,-
1.0644334922311949E7,6261709.751605329,EPSG:3857)
Width / # Columns: 3
Height / # Rows: 2
MinX: 27, MinY: 82, MaxX: 29, MaxY: 83

Tiles:
Tile(x:27, y:82, z:7)
Tile(x:28, y:82, z:7)
Tile(x:29, y:82, z:7)
Tile(x:27, y:83, z:7)
Tile(x:28, y:83, z:7)
Tile(x:29, y:83, z:7)
```

# OSM

Create a TileLayer for OSM tiles.

```
OSM osm = new OSM()
```

Create a TileLayer for OSM tiles with custom urls.

```
OSM osm = new OSM("OSM", [
    "http://a.tile.openstreetmap.org",
    "http://b.tile.openstreetmap.org",
    "http://c.tile.openstreetmap.org"
])
```

### Standard OSM

Create a TileLayer for OSM tiles.

```
OSM osm = OSM.getWellKnownOSM("osm")
```

### Stamen Toner

Create a TileLayer for OSM Stamen Toner tiles.

```
OSM osm = OSM.getWellKnownOSM("stamen-toner")
```

### Stamen Toner Lite

Create a TileLayer for OSM Stamen Toner Lite tiles.

```
OSM osm = OSM.getWellKnownOSM("stamen-toner-lite")
```

### Stamen Water Color

Create a TileLayer for OSM Stamen Water Color tiles.

```
OSM osm = OSM.getWellKnownOSM("stamen-watercolor")
```

### Stamen Terrain

Create a TileLayer for OSM Stamen Terrain tiles.

```
OSM osm = OSM.getWellKnownOSM("stamen-terrain")
```

North America
Europe
Asia
Africa
South America
Australia
Antarctica