

Table of Contents

| | |
|---|----|
| Style Recipes | 1 |
| Creating Basic Styles | 1 |
| Creating Strokes | 4 |
| Creating Fills | 7 |
| Creating Shapes | 11 |
| Creating Icons | 13 |
| Creating Labels | 14 |
| Creating Transforms | 17 |
| Creating Gradients | 18 |
| Creating Unique Values | 20 |
| Creating Color Maps | 22 |
| Creating Channel Selection and Contrast Enhancement | 24 |
| Reading and Writing Styles | 26 |

Style Recipes

Styles are found in the [geoscript.style](#) package.

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

Creating Basic Styles

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



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

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



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

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



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

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



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

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





Creating Strokes

Create a Stroke Symbolizer with a Color

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



Create a Stroke Symbolizer with a Color and Width

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



Create a Stroke Symbolizer with casing

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



Create a Stroke Symbolizer with Dashes

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



Create a Stroke Symbolizer with railroad Hatching

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



Create a Stroke Symbolizer with spaced Shape symbols

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



Create a Stroke Symbolizer with alternating spaced Shape symbols

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



Creating Fills

Create a Fill Symbolizer with a Color

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




Create a Fill Symbolizer with a Color and a Stroke

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



Create a Fill Symbolizer with a Color and Opacity

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



Create a Fill Symbolizer from named parameters

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



Create a Fill Symbolizer with an Icon

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



Create a Fill Symbolizer with a Hatch

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



Create a Fill Symbolizer with a random fill

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



Creating Shapes

Create a Shape Symbolizer with a Color

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



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

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



Create a Shape Symbolizer with named parameters

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



Create a Shape Symbolizer with Stroke outline

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



Creating Icons

Create an Icon Symbolizer

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



Create an Icon Symbolizer

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



Creating Labels

Create a Label for a Point Layer

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

- ① anchor
- ② displacement
- ③ rotation



Create a Label for a Point Layer with a Font

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

- ① style (normal, italic, oblique)
- ② weight (normal, bold)
- ③ size (8,12,16,ect..)
- ④ family (serif, arial, verdana)



Create a Label for a Point Layer with Halo

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




Create a Label for a Polygon Layer

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



Create a Label for a Line Layer

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



Creating Transforms

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

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



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

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



Creating Gradients

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

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



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

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



Create a custom Gradient Symbolizer between Symbolizers and values

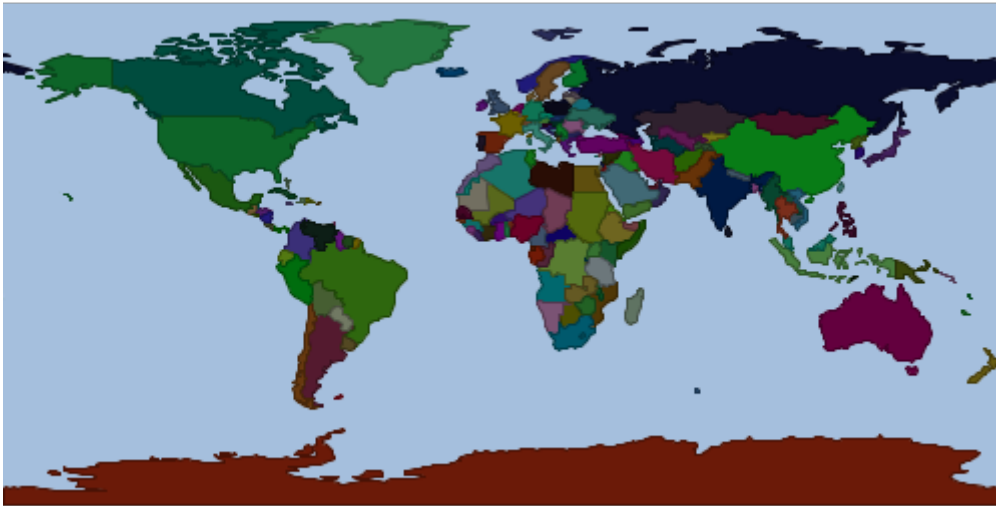
```
Gradient gradient = new Gradient(  
    new Property("POP2020"),  
    [0, 10000, 20000, 30000],  
    [  
        new Shape("white", 4).stroke("black", 0.5),  
        new Shape("#b0d2e8", 8).stroke("black", 0.5),  
        new Shape("#3e8ec4", 16).stroke("black", 0.5),  
        new Shape("#08306b",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



Creating Unique Values

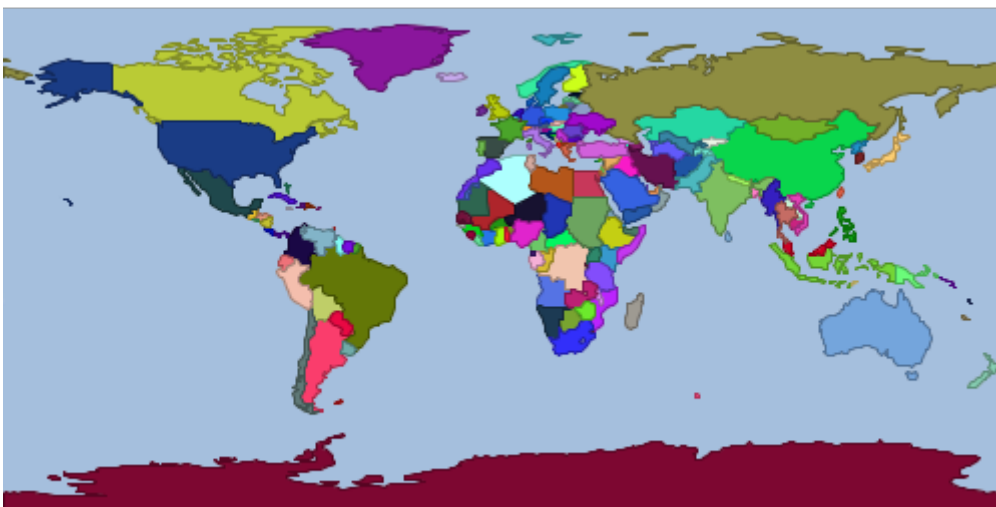
Create a Unique Values Symbolizer from a Layer's Field

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



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

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



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

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



Creating Color Maps

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

```
Format format = new GeoTIFF(new File('src/main/resources/pc.tif'))
Raster raster = format.read()
ColorMap colorMap = new ColorMap([
    [color: "#9fd182", quantity:25],
    [color: "#3e7f3c", quantity:470],
    [color: "#133912", quantity:920],
    [color: "#08306b", quantity:1370],
    [color: "#ffffff5", 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: "#ffffff5", 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: "#ffffff5", 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
```

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>
```

```
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 ysl = writer.write(symbolizer)
println ysl
```

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

```
String yslld = ""
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
```

