Table of Contents

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 ReferencedEnvelope[-102.0 : -100.0, 43.5 : 47.5] ]
```

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 ]
```

Create an equals Filter

```
Filter filter = Filter.equals("name", "Washington")
println filter.toString()
```

```
[ name = Washington ]
```

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
}
```

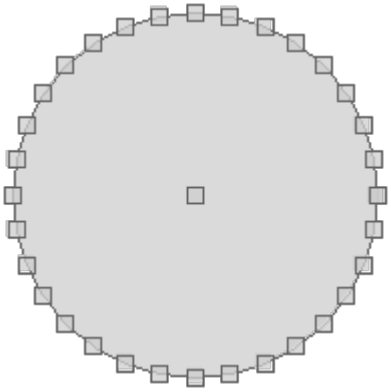
```
There are 318 Functions:
And
Area
Categorize
Collection_Average
Collection_Bounds
Collection_Count
Collection_Max
Collection_Median
Collection_Min
Collection_Nearest
```

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
```

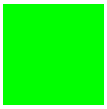
Create a ProcessFunction from a Process that converts geometries in a Layer into a bounds.

```
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 RGB function string

```
Color color = new Color("rgb(0,128,128)")
```

Create a Color from a HLS Map

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

Create a Color from a HSL function string

```
Color color = new Color("hsl(0,1,0.5)")
```

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]
```

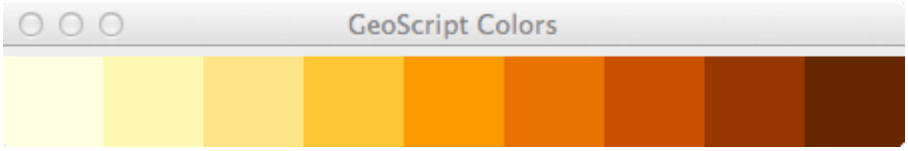
Displaying Colors

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
}
```

```
PiYG
RdGy
Purples
Reds
BuPu
Set1
PuBuGn
Grays
PuOr
Accents
YlGn
YlOrBr
RdPu
PuRd
RdYlGn
Paired
Set3
Set2
GnBu
YlGnBu
RdYlBu
RdBu
BuGn
BrBG
PRGn
Blues
Greens
OrRd
Dark2
Oranges
Spectral
Pastel2
Pastel1
PuBu
YlOrRd
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
}
```

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

Get sequential color palettes

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

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

Get qualitative color palettes

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

```
Set1
Accents
Paired
Set3
Set2
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)")
```