

Table of Contents

Filter Recipes	1
Creating Filters	1
Using Filters	1
Evaluating Filters	2
Creating Literals	3
Creating Properties	4
Evaluating Properties	4
Creating Functions	5
Evaluating Functions	6
Creating Colors	7
Getting Color Formats	8
Displaying Colors	9
Using Color Palettes	10
Creating Expressions from CQL	15

Filter Recipes

Creating Filters

Create a Filter from a CQL string

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

```
[ name = Seattle ]
```

Create a Filter from a CQL string

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

```
[ soilType = Mollisol ]
```

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>
```

Evaluating Filters

Test to see if a Filter matches a Feature by attribute

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

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

true

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

false

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

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

true

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

false

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

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

true

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

false

Creating Literals

Create a literal Expression from a number

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

3.56

Create a literal Expression from a string

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

Seattle

Evaluating a literal Expression just gives you the value

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

3.56

Creating Properties

Create a Property from a string

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

name

Create a Property from a Field

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

geom

Evaluating Properties

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

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

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

1

Get the name

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

Seattle

Get the geometry

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

POINT (-122.3204 47.6024)

Creating Functions

Create a Function from a CQL string

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

centroid([the_geom])

Create a Function from a name and Expressions

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

centroid([the_geom])

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

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

my_centroid([the_geom])

Create a Function from a CQL string and a Closure

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

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

You can get a list of built in Functions

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

There are 277 Functions:

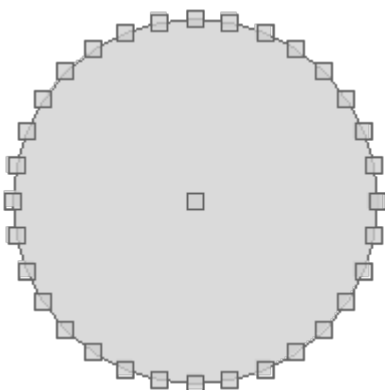
Area
Categorize
Collection_Average
Collection_Bounds
Collection_Count
Collection_Max
Collection_Median
Collection_Min
Collection_Nearest
Collection_Sum

Evaluating Functions

Evaluate 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)
```



Evaluate a geometry Function

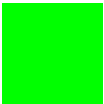
```
Function lowerCaseFunction = new Function("strToLowerCase(name)")
String lowerCaseName = lowerCaseFunction.evaluate(feature)
println lowerCaseName
```

seattle

Creating Colors

Create a Color from a RGB color string

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



Create a Color from a CSS color name

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



Create a Color from a hexadecimal string

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



Create a Color from a RGB List

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



Create a Color from a RGB Map

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




Create a Color from a HLS Map

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



Get a Random Color

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



Get a Random Pastel Color

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



Get a darker Color

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



Get a brighter Color

```
Color color = new Color("purple")  
Color brighterColor = 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

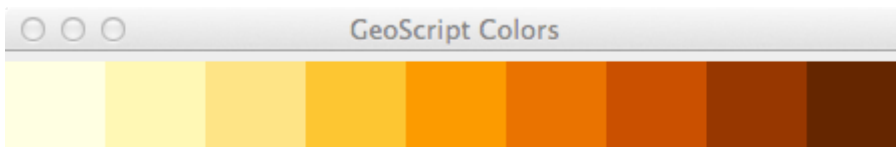
Draw a List of Colors to a BufferedImage

```
Color color = new Color("pink")
BufferedImage image = Color.drawImage(
    [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")
```



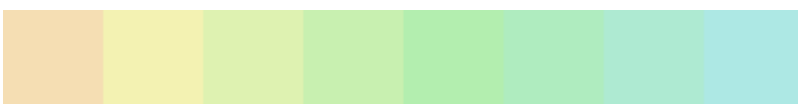
Create a Color palette by interpolating between two colors

```
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

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

NAME

Create a Function from a CQL String

```
Expression expression = Expression.fromCQL("centroid(the_geom)")  
println expression
```

centroid([the_geom])