

# Table of Contents

Process Recipes .....	1
Execute a built-in Process .....	1
Listing built-in Processes .....	2
Executing a new Process .....	5
Process Functions.....	6

# 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]
```

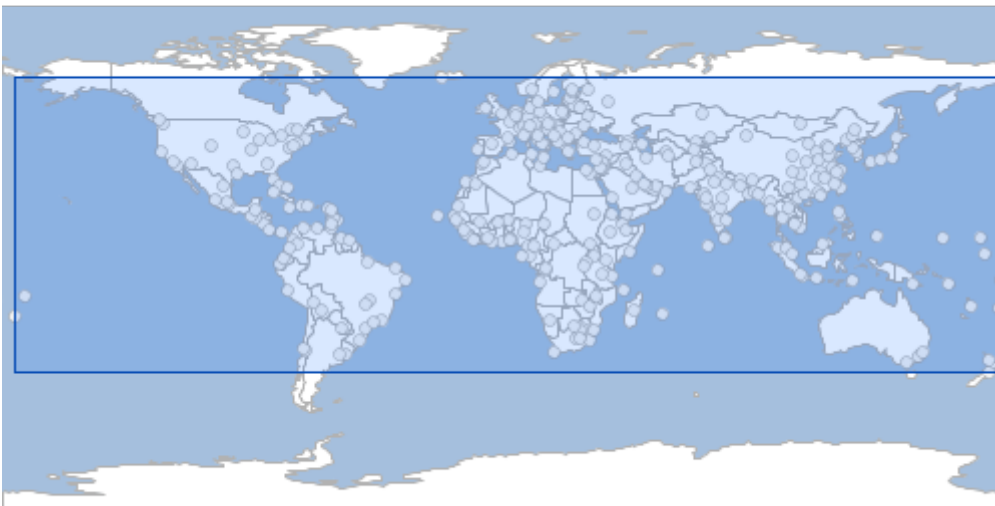
*Get the output parameters*

```
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:LRSegment  
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:length  
geo:isEmpty  
geo:contains  
geo:disjoint  
geo:intersects  
geo:area  
geo:numGeometries  
geo:isClosed  
geo:buffer  
geo:difference  
geo:boundary  
geo:splitPolygon  
geo:within  
geo:touches  
geo:convexHull  
geo:crosses  
geo:symDifference  
geo:distance  
geo:centroid  
geo:interiorPoint  
geo:getGeometryN  
geo:overlaps  
geo:isSimple  
geo:isWithinDistance

geo:relate  
geo:densify  
geo:simplify  
geo:startPoint  
geo:numPoints  
geo:reproject  
geo:dimension  
geo:exteriorRing  
geo:numInteriorRing  
geo:geometryType  
geo:envelope  
geo:isRing  
geo:polygonize  
geo:endPoint  
geo>equalsExact  
geo:isValid  
geo:union  
geo:intersection  
geo:getX  
geo:getY  
geo:pointN  
geo:interiorRingN  
geo:relatePattern  
geo>equalsExactTolerance  
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  
geoscript:bounds

# 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
```

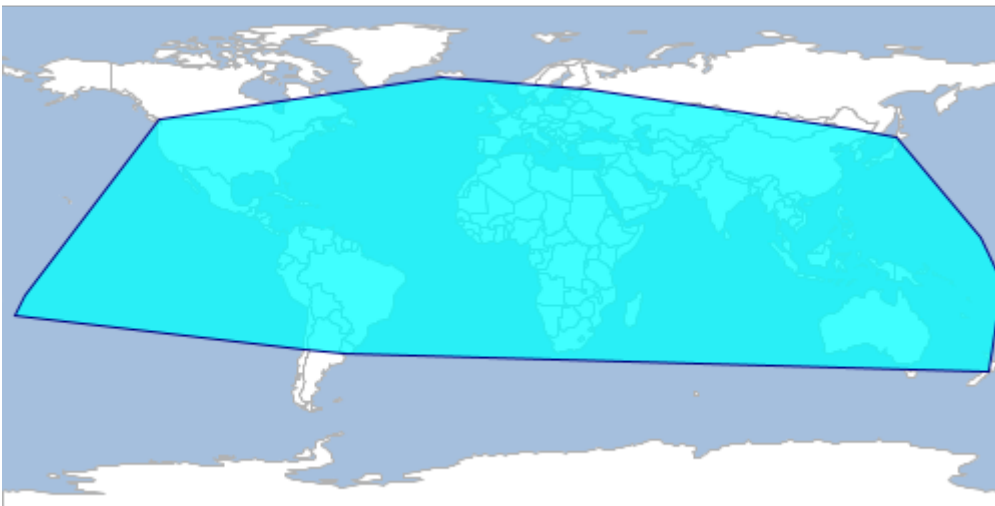


## Process Functions

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
```

