

# Table of Contents

Tile Recipes .....	1
Tile .....	1
Grid .....	2
Pyramid .....	2
Generating Tiles .....	15
Tile Layer .....	23
TileRenderer .....	40
TileCursor .....	44
TMS .....	51
MBTiles .....	52
DBTiles .....	53
GeoPackage .....	54
UTFGrid .....	56
VectorTiles .....	56
Generating TileLayer .....	58
OSM .....	59

# 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 = iVBORw0KGgoAAAANSUhEUgAAQAAAAEACAYAAABccqhmAAAtOU...
```

### 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, 2.0037471205137067E7, 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,EP
SG: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,EP
SG: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}"
```

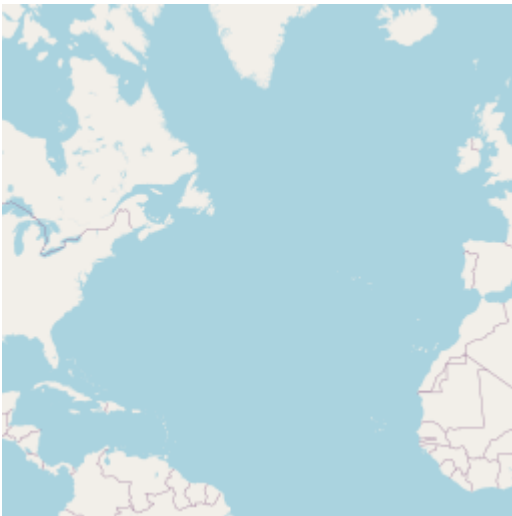
```
Projection: EPSG:3857
Origin: BOTTOM_LEFT
Bounds: (-2.0036395147881314E7,-
2.0037471205137067E7,2.0036395147881314E7,2.003747120513706E7,EP
SG:3857)
Max Zoom: 19
```

## 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,EPSSG: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", "EPSSG: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,EPSSG: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](https://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
```

## Find a Pyramid Reader

```
PyramidReader reader = PyramidReaders.find("csv")
Pyramid pyramid = reader.read("""EPSG:3857
-2.0036395147881314E7,
-2.0037471205137067E7,2.0036395147881314E7,2.0037471205137067E7,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,-
2.0037471205137067E7,2.0036395147881314E7,2.0037471205137067E7,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": 2.0037471205137067E7
  },
  "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>2.0037471205137067E7</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,
-2.0037471205137067E7,2.0036395147881314E7,2.0037471205137067E7,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>2.0037471205137067E7</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>
```

```
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>-2.003747120513706E7</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,-
2.003747120513706E7,2.0036395147881314E7,2.003747120513706E7, 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 MBTiles file with metatiles

```

File file = new File("target/world_meta.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, metatile: [width:4, height: 4])

```

[tile generate mbtiles metatile] | *tile\_generate\_mbtiles\_metatile.png*

## DBTiles

Generate Image Tiles to a MBTiles like JDBC Database.

```

File file = new File("target/world_tiles.db")
DBTiles dbtiles = new DBTiles("jdbc:h2:${file}", "org.h2.Driver", "World", "World wide
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(dbtiles, [ocean, countries])
TileGenerator generator = new TileGenerator()
generator.generate(dbtiles, renderer, 0, 2)

```



## GeoPackage

Generate Image Tiles to a GeoPackage file

```
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)
```



## TMS

Generate Image Tiles to a TMS directory

```
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)
```



## UTFGrid

Generate UTFGrid tiles to a directory

```
File directory = new File("target/utfgrid")
directory.mkdir()
UTFGrid utf = new UTFGrid(directory)

Workspace workspace = new GeoPackage('src/main/resources/data.gpkg')
Layer countries = workspace.get("countries")

UTFGridTileRenderer renderer = new UTFGridTileRenderer(utf, countries, [countries
.schema.get("NAME")])
TileGenerator generator = new TileGenerator()
generator.generate(utf, renderer, 0, 2)
```



## Vector Tiles

Generate vector tiles to a directory

```

File directory = new File("target/pbf")
directory.mkdir()

Workspace workspace = new Directory("src/main/resources/shapefiles")
Layer countries = workspace.get("countries")
Layer ocean = workspace.get("ocean")

Pyramid pyramid = Pyramid.createGlobalMercatorPyramid()
pyramid.origin = Pyramid.Origin.TOP_LEFT
VectorTiles vectorTiles = new VectorTiles(
    "world",
    directory,
    pyramid,
    "pbf",
    style: [
        "countries": new Fill("white") + new Stroke("black", 1),
        "ocean": new Fill("blue")
    ]
)

PbfVectorTileRenderer renderer = new PbfVectorTileRenderer([countries, ocean], [
    "countries": ["NAME"],
    "ocean": ["FeatureCla"]
])
TileGenerator generator = new TileGenerator()
generator.generate(vectorTiles, renderer, 0, 2)

```



Generate vector tiles to a MBTiles file

```

File file = new File("target/vectortiles.mbtiles")

Workspace workspace = new Directory("src/main/resources/shapefiles")
Layer countries = workspace.get("countries")
Layer ocean = workspace.get("ocean")

Pyramid pyramid = Pyramid.createGlobalMercatorPyramid()
pyramid.origin = Pyramid.Origin.TOP_LEFT
VectorTiles vectorTiles = new VectorTiles(
    "world",
    file,
    pyramid,
    "pbf",
    style: [
        "countries": new Fill("white") + new Stroke("black", 1),
        "ocean": new Fill("blue")
    ]
)

PbfVectorTileRenderer renderer = new PbfVectorTileRenderer([countries, ocean], [
    "countries": ["NAME"],
    "ocean": ["FeatureCla"]
])
TileGenerator generator = new TileGenerator()
generator.generate(vectorTiles, renderer, 0, 2)

```



## Tile Layer



## Tile Layer Properties

Create a TileLayer from an MBTiles File.

```
File file = new File("src/main/resources/tiles.mbtiles")
MTiles mbtiles = new MTiles(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,-
2.0037471205137067E7,2.0036395147881314E7,2.0037471205137067E7,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,-
2.0037471205137067E7,2.0036395147881314E7,2.0037471205137067E7,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

## Using Tile Layers

Get a TileLayer and make sure it is closed when done.

```
File file = new File("src/main/resources/tiles.mbtiles")
TileLayer.withTileLayer(new MBTiles(file)) { TileLayer tileLayer ->
    println tileLayer.name
    println tileLayer.proj
    println tileLayer.bounds
}
```

```
countries
EPSG:3857
(-2.0036395147881314E7,-
2.0037471205137067E7,2.0036395147881314E7,2.0037471205137067E7,EPSG:3857)
```

## Get Tile Layer from a Map

Get an MBTiles TileLayer from a Map

```
TileLayer mbtiles = TileLayer.getTileLayer([type:'mbtiles', file:
'src/main/resources/tiles.mbtiles'])
println "${mbtiles.name} ${mbtiles.proj} ${mbtiles.bounds} ${mbtiles.pyramid}"
```

```
countries EPSG:3857 (-2.0036395147881314E7,-
2.0037471205137067E7,2.0036395147881314E7,2.0037471205137067E7,EPSG:3857)
geoscript.layer.Pyramid(proj:EPSG:3857, bounds:(-2.0036395147881314E7,-
2.0037471205137067E7,2.0036395147881314E7,2.0037471205137067E7,EPSG:3857),
origin:BOTTOM_LEFT, tileWidth:256, tileHeight:256)
```

Get an GeoPackage TileLayer from a Map

```
TileLayer geopackage = TileLayer.getTileLayer([type: 'geopackage', name: 'world',
file: 'src/main/resources/tiles.gpkg'])
println "${geopackage.name} ${geopackage.proj} ${geopackage.bounds}
${geopackage.pyramid}"
```

```
world EPSG:4326 (-179.99,-89.99,179.99,89.99,EPG:4326)
geoscript.layer.Pyramid(proj:EPSG:4326, bounds:(-179.99,-
89.99,179.99,89.99,EPG:4326), origin:TOP_LEFT, tileWidth:256, tileHeight:256)
```

Get an TMS TileLayer from a Map

```
TileLayer tms = TileLayer.getTileLayer([type: 'tms', file: 'src/main/resources/tms',
format: 'png', pyramid: 'globalmercator'])
println "${tms.name} ${tms.proj} ${tms.bounds} ${tms.pyramid}"
```

```
tms EPSG:3857 (-2.0036395147881314E7,-
2.0037471205137067E7,2.0036395147881314E7,2.0037471205137067E7,EPG:3857)
geoscript.layer.Pyramid(proj:EPSG:3857, bounds:(-2.0036395147881314E7,-
2.0037471205137067E7,2.0036395147881314E7,2.0037471205137067E7,EPG:3857),
origin:BOTTOM_LEFT, tileWidth:256, tileHeight:256)
```

Get an OSM TileLayer from a Map

```
TileLayer osm = TileLayer.getTileLayer([type: 'osm', url:
'http://a.tile.openstreetmap.org'])
println "${osm.name} ${osm.proj} ${osm.bounds} ${osm.pyramid}"
```

```
OSM EPSG:3857 (-2.0036395147881314E7,-
2.0037471205137067E7,2.0036395147881314E7,2.0037471205137067E7,EPG:3857)
geoscript.layer.Pyramid(proj:EPSG:3857, bounds:(-2.0036395147881314E7,-
2.0037471205137067E7,2.0036395147881314E7,2.0037471205137067E7,EPG:3857),
origin:TOP_LEFT, tileWidth:256, tileHeight:256)
```

Get an PBF Vector TileLayer from a Map

```
TileLayer pbf = TileLayer.getTileLayer([type: 'vectortiles', name: 'world', file:
'src/main/resources/pbf', format: 'pbf', pyramid: 'GlobalMercator'])
println "${pbf.name} ${pbf.proj} ${pbf.bounds} ${pbf.pyramid}"
```

```
world EPSG:3857 (-2.0036395147881314E7,-
2.0037471205137067E7,2.0036395147881314E7,2.0037471205137067E7,EPG:3857)
geoscript.layer.Pyramid(proj:EPSG:3857, bounds:(-2.0036395147881314E7,-
2.0037471205137067E7,2.0036395147881314E7,2.0037471205137067E7,EPG:3857),
origin:BOTTOM_LEFT, tileWidth:256, tileHeight:256)
```

Get an UTF TileLayer from a Map

```
TileLayer utf = TileLayer.getTileLayer([type: 'utfgrid', file:
'src/main/resources/utf'])
println "${utf.name} ${utf.proj} ${utf.bounds} ${utf.pyramid}"
```

```
utf EPSG:3857 (-2.0036395147881314E7,-
2.0037471205137067E7,2.0036395147881314E7,2.0037471205137067E7,EP
SG:3857)
geoscript.layer.Pyramid(proj:EPSG:3857, bounds:(-2.0036395147881314E7,-
2.0037471205137067E7,2.0036395147881314E7,2.0037471205137067E7,EP
SG:3857),
origin:TOP_LEFT, tileWidth:256, tileHeight:256)
```

## Get Tile Layer from a String

Get an MBTiles TileLayer from a String

```
TileLayer mbtiles = TileLayer.getTileLayer("type=mbtiles
file=src/main/resources/tiles.mbtiles")
println "${mbtiles.name} ${mbtiles.proj} ${mbtiles.bounds} ${mbtiles.pyramid}"
```

```
countries EPSG:3857 (-2.0036395147881314E7,-
2.0037471205137067E7,2.0036395147881314E7,2.0037471205137067E7,EP
SG:3857)
geoscript.layer.Pyramid(proj:EPSG:3857, bounds:(-2.0036395147881314E7,-
2.0037471205137067E7,2.0036395147881314E7,2.0037471205137067E7,EP
SG:3857),
origin:BOTTOM_LEFT, tileWidth:256, tileHeight:256)
```

Get an GeoPackage TileLayer from a String

```
TileLayer geopackage = TileLayer.getTileLayer("type=geopackage name=world
file=src/main/resources/tiles.gpkg")
println "${geopackage.name} ${geopackage.proj} ${geopackage.bounds}
${geopackage.pyramid}"
```

```
world EPSG:4326 (-179.99,-89.99,179.99,89.99,EP
SG:4326)
geoscript.layer.Pyramid(proj:EPSG:4326, bounds:(-179.99,-
89.99,179.99,89.99,EP
SG:4326), origin:TOP_LEFT, tileWidth:256, tileHeight:256)
```

Get an TMS TileLayer from a String

```
TileLayer tms = TileLayer.getTileLayer("type=tms file=src/main/resources/tms
format=png pyramid=globalmercator")
println "${tms.name} ${tms.proj} ${tms.bounds} ${tms.pyramid}"
```

```
tms EPSG:3857 (-2.0036395147881314E7,-  
2.0037471205137067E7,2.0036395147881314E7,2.0037471205137067E7,EP  
SG:3857)  
geoscript.layer.Pyramid(proj:EPSG:3857, bounds:(-2.0036395147881314E7,-  
2.0037471205137067E7,2.0036395147881314E7,2.0037471205137067E7,EP  
SG:3857),  
origin:BOTTOM_LEFT, tileWidth:256, tileHeight:256)
```

Get an OSM TileLayer from a String

```
TileLayer osm = TileLayer.getTileLayer("type=osm url=http://a.tile.openstreetmap.org")  
println "${osm.name} ${osm.proj} ${osm.bounds} ${osm.pyramid}"
```

```
OSM EPSG:3857 (-2.0036395147881314E7,-  
2.0037471205137067E7,2.0036395147881314E7,2.0037471205137067E7,EP  
SG:3857)  
geoscript.layer.Pyramid(proj:EPSG:3857, bounds:(-2.0036395147881314E7,-  
2.0037471205137067E7,2.0036395147881314E7,2.0037471205137067E7,EP  
SG:3857),  
origin:TOP_LEFT, tileWidth:256, tileHeight:256)
```

Get an PBF Vector TileLayer from a String

```
TileLayer pbf = TileLayer.getTileLayer("type=vectortiles name=world  
file=src/main/resources/pbf format=pbf pyramid=GlobalMercator")  
println "${pbf.name} ${pbf.proj} ${pbf.bounds} ${pbf.pyramid}"
```

```
world EPSG:3857 (-2.0036395147881314E7,-  
2.0037471205137067E7,2.0036395147881314E7,2.0037471205137067E7,EP  
SG:3857)  
geoscript.layer.Pyramid(proj:EPSG:3857, bounds:(-2.0036395147881314E7,-  
2.0037471205137067E7,2.0036395147881314E7,2.0037471205137067E7,EP  
SG:3857),  
origin:BOTTOM_LEFT, tileWidth:256, tileHeight:256)
```

Get an UTF TileLayer from a String

```
TileLayer utf = TileLayer.getTileLayer("type=utfgrid file=src/main/resources/utf")  
println "${utf.name} ${utf.proj} ${utf.bounds} ${utf.pyramid}"
```

```
utf EPSG:3857 (-2.0036395147881314E7,-  
2.0037471205137067E7,2.0036395147881314E7,2.0037471205137067E7,EP  
SG:3857)  
geoscript.layer.Pyramid(proj:EPSG:3857, bounds:(-2.0036395147881314E7,-  
2.0037471205137067E7,2.0036395147881314E7,2.0037471205137067E7,EP  
SG:3857),  
origin:TOP_LEFT, tileWidth:256, tileHeight:256)
```



# TileRenderer

TileRenderers know how to create a Tile for a given Bounds. GeoScript has TileRenderer for creating images, vector tiles, and utfgrids.

## Get default TileRenderer

Get a default TileRenderer for a TileLayer

```
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("#a5bfd9")

TileLayer tileLayer = TileLayer.getTileLayer([type:'mbtiles', file:
'target/countries.mbtiles'])
TileRenderer tileRenderer = TileLayer.getTileRenderer(tileLayer, [ocean, countries])
Pyramid pyramid = tileLayer.pyramid
Tile tile = tileLayer.get(0,0,0)
Bounds bounds = pyramid.bounds(tile)
tile.data = tileRenderer.render(bounds)
tileLayer.put(tile)
```



## ImageTileRenderer

Use an ImageTileRenderer to create an image Tile.

```

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("#a5bfd9")

TileLayer tileLayer = TileLayer.getTileLayer([type:'mbtiles', file:
'target/countries.mbtiles'])
ImageTileRenderer tileRenderer = new ImageTileRenderer(tileLayer, [ocean, countries])
Pyramid pyramid = tileLayer.pyramid
Tile tile = tileLayer.get(0,0,0)
Bounds bounds = pyramid.bounds(tile)
tile.data = tileRenderer.render(bounds)
tileLayer.put(tile)

```



## VectorTileRenderer

Use an VectorTileRenderer to create a Vector Tile.

```

Workspace workspace = new Directory("src/main/resources/shapefiles")
Layer countries = workspace.get("countries")

File directory = new File("target/country_geojson_tiles")
directory.mkdir()

Pyramid pyramid = Pyramid.createGlobalMercatorPyramid()
VectorTiles tileLayer = new VectorTiles(
    "countries",
    directory,
    pyramid,
    "geojson"
)

GeoJSONWriter writer = new GeoJSONWriter()
VectorTileRenderer tileRenderer = new VectorTileRenderer(writer, countries, [
    countries.schema.get("NAME")])
Tile tile = tileLayer.get(0,0,0)
Bounds bounds = pyramid.bounds(tile)
tile.data = tileRenderer.render(bounds)
tileLayer.put(tile)

```

```

{"type":"FeatureCollection","features":[{"type":"Feature","geometry":{"type":"MultiPolygon","coordinates":[[[[[180,-16.0671],[180,-16.5552],[179.3641,-16.8014],[178.7251,-17.012],[178.5968,-16.6392],[179.0966,-16.434],[179.4135,-16.3791],[180,-16.0671]]],[[[178.1256,-17.5048],[178.3736,-17.3399],[178.7181,-17.6285],[178.5527,-18.1506],[177.9327,-18.288],[177.3815,-18.1643],[177.285,-17.7246],[177.6709,-17.3811],[178.1256,-17.5048]]],[[[-179.7933,-16.0209],[-179.9174,-16.5018],[-180,-16.5552],[-180,-16.0671],[-179.7933,-16.0209]]]]}},{"properties":{"NAME":"Fiji"},"id":"fid--4c337b6e_1722f8b0815_-219a"}, {"type":"Feature","geometry":{"type":"MultiPolygon","coordinates":[[[[[33.9037,-0.95],[34.0726,-1.0598],[37.6987,-3.097],[37.7669,-3.6771],[39.2022,-4.6768],[38.7405,-5.9089],[38.7998,-6.4757],[39.44,-6.84],[39.47,-7.1],[39.1947,-7.7039],[39.252,-8.0078],[39.1865,-8.4855],[39.5357,-9.1124],[39.

```

## PbfVectorTileRenderer

Use an PbfVectorTileRenderer to create a Vector Tile.

```

Workspace workspace = new Directory("src/main/resources/shapefiles")
Layer countries = workspace.get("countries")

File directory = new File("target/country_pbf_tiles")
directory.mkdir()

Pyramid pyramid = Pyramid.createGlobalMercatorPyramid()
pyramid.origin = Pyramid.Origin.TOP_LEFT
VectorTiles tileLayer = new VectorTiles(
    "countries",
    directory,
    pyramid,
    "pbf"
)

PbfVectorTileRenderer tileRenderer = new PbfVectorTileRenderer(countries, [countries
.schema.get("NAME")])
Tile tile = tileLayer.get(0,0,0)
Bounds bounds = pyramid.bounds(tile)
tile.data = tileRenderer.render(bounds)
tileLayer.put(tile)

```

## UTFGridTileRenderer

Use an UTFGridTileRenderer to create a UTFGrid Tile.

```

Workspace workspace = new GeoPackage('src/main/resources/data.gpkg')
Layer countries = workspace.get("countries")

File directory = new File("target/countryUtfGrid")
directory.mkdir()
UTFGrid tileLayer = new UTFGrid(directory)

UTFGridTileRenderer tileRenderer = new UTFGridTileRenderer(tileLayer, countries,
[countries.schema.get("NAME")])
Pyramid pyramid = tileLayer.pyramid
Tile tile = tileLayer.get(0,0,0)
Bounds bounds = pyramid.bounds(tile)
tile.data = tileRenderer.render(bounds)
tileLayer.put(tile)

```

```
{
  "grid": [
    "",
    "",
    "",
    "",
    "      !      ###",
    "        !!!!!   #####",
    "          !!!!!#####",
    "            !!!!!#####",
    "              !!!!!#####    $$$",
    "                !!!!!#####    %%%    $",
    "                  !   !!! #####    %%    $$",
    "                    ! !! !!! #####    %%    $",
    "                     !   !!!!! #####    %    $    $$",
    "                      !!!!!!!   #####
```

# 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,-
2.0037471205137067E7,2.0036395147881314E7,2.0037471205137067E7,EP
SG: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)

```

## TMS

Access a TileLayer from an TMS directory

```
File dir = new File("src/main/resources/tms")
Pyramid pyramid = Pyramid.createGlobalMercatorPyramid()
TMS tms = new TMS(
    "world", ①
    "png", ②
    dir, ③
    pyramid ④
)
```

- ① Name
- ② Image type
- ③ Directory
- ④ Pyramid



## MBTiles

Access a TileLayer from an MBTiles file

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



## DBTiles

Access a TileLayer from an DBTiles H2 database

```
File file = new File("src/main/resources/h2dbtiles/world_tiles.db")
DBTiles dbtiles = new DBTiles("jdbc:h2:${file}", "org.h2.Driver", "World", "World wide
tiles")
```



## GeoPackage

Access a TileLayer with a Global Geodetic Pyramid from an GeoPackage file

```
File file = new File("src/main/resources/data.gpkg")
geoscript.layer.GeoPackage geopackage = new geoscript.layer.GeoPackage(file, "world")
```



Access a `TileLayer` with a Global Mercator Pyramid from an `GeoPackage` file

```
File file = new File("src/main/resources/data.gpkg")
geoscript.layer.GeoPackage geopackage = new geoscript.layer.GeoPackage(file,
"world_mercator")
```





# UTFGrid

Access a TileLayer from an UTFGrid directory

```
File dir = new File("src/main/resources/utf")
UTFGrid utfGrid = new UTFGrid(dir)
```

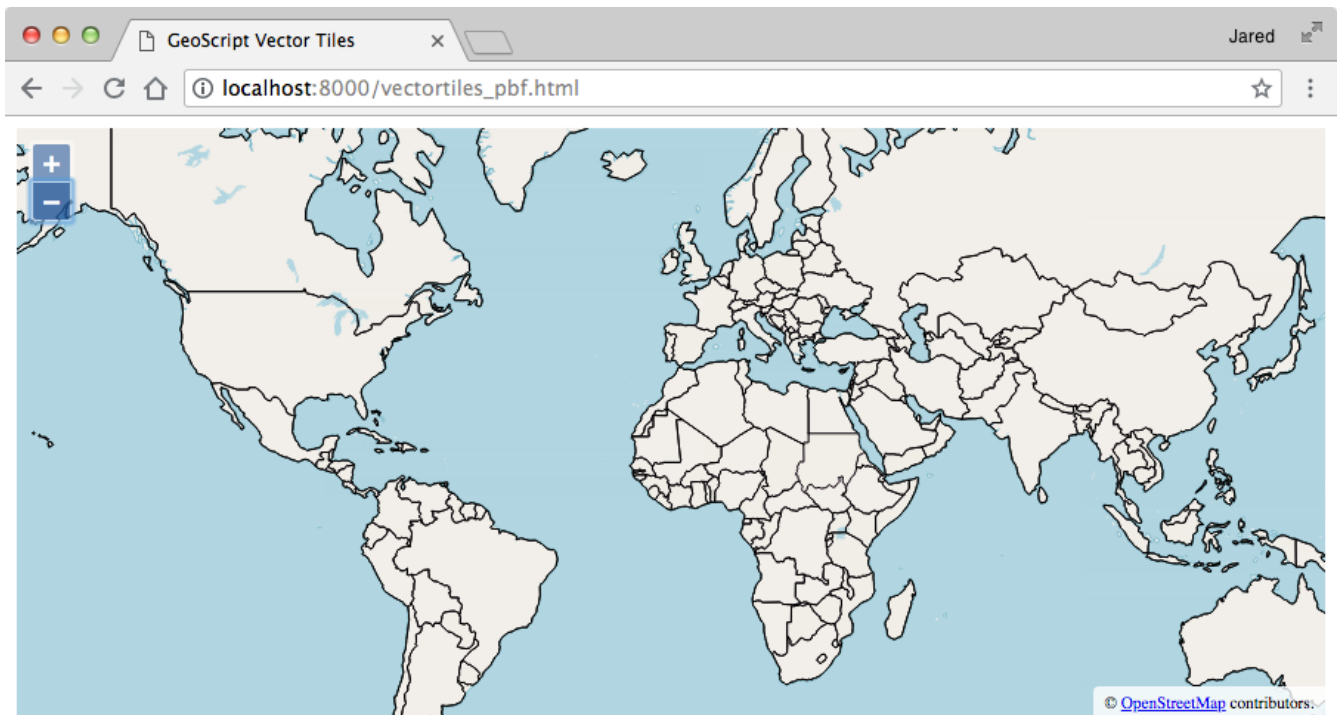


# VectorTiles

Access a TileLayer from an VectorTiles directory

```
File dir = new File("src/main/resources/pbf")
Pyramid pyramid = Pyramid.createGlobalMercatorPyramid()
VectorTiles vectorTiles = new VectorTiles(
    "World", ①
    dir,      ②
    pyramid,  ③
    "pbf"     ④
)
```

- ① Name
- ② Directory
- ③ Pyramid
- ④ Type



Access a `TileLayer` from an `VectorTiles` `MBTiles` file

```
File file = new File("src/main/resources/vectortiles.mbtiles")
Pyramid pyramid = Pyramid.createGlobalMercatorPyramid()
VectorTiles vectorTiles = new VectorTiles(
    "World", ①
    file,    ②
    pyramid, ③
    "pbf"    ④
)
```

- ① Name
- ② MBTiles File
- ③ Pyramid
- ④ Type



## Generating TileLayer

A `GeneratingTileLayer` can create tiles on demand when they are accessed.

```
File dir = new File("target/worldtiles")
Pyramid pyramid = Pyramid.createGlobalMercatorPyramid()
TileLayer tileLayer = new TMS("World", "png", dir, pyramid)

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")
TileRenderer tileRenderer = new ImageTileRenderer(tileLayer, [ocean, countries])

GeneratingTileLayer generatingTileLayer = new GeneratingTileLayer(tileLayer,
    tileRenderer)
Tile tile = generatingTileLayer.get(0, 0, 0)
```



## 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")
```

