

Table of Contents

| | |
|-----------------------|----|
| Tile Recipes | 1 |
| Tile..... | 1 |
| Grid | 2 |
| Pyramid | 2 |
| Generating Tiles..... | 12 |
| Tile Layer..... | 15 |
| TileCursor | 17 |

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 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
```

Reading and Writing Pyramids

The Pyramid IO classes are in the 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.className.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": 2.003747120513706E7
  },
  "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>
  </grids>
</pyramid>

```

```

</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>2.003747120513706E7</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>

```

Read a Pyramid from a GDAL XML File

```

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>-2.0037471205137067E7</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>'''

GdalTmsPyramidReader reader = new GdalTmsPyramidReader()
Pyramid pyramid = reader.read(xml)

```

```
geoscript.layer.Pyramid(proj:EPSG:3857, bounds:(-2.0036395147881314E7,-  
2.0037471205137067E7,2.0036395147881314E7,2.003747120513706E7,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 GeoPackage file

GeoPackage

```
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)
```



Generate Image Tiles to a TMS directory

TMS

```
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)
```



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,-
2.0037471205137067E7,2.0036395147881314E7,2.003747120513706E7,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,-
2.0037471205137067E7,2.0036395147881314E7,2.003747120513706E7,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)
```



TileCursor

A TileCursor is a way to get a collection of Tiles from a TileLayer.

Get a TileCursor with all of the Tiles form 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 form 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 form 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)

```