GEOS Documentation

Release 0.2.0

Gregor Sturm

Feb 21, 2021

Contents

This is a python-based server for creating Google Earth overlays of tiled maps. Your can now also display maps in the web browser, measure distances and print maps as high-quality PDF's.

Developers can also use *GEOS* as a library to convert coordinates or to access the map printing functionality programmatically.

CHAPTER 1

Installation

Instructions for running GEOS in a docker container can be found *below*.

1.1 Requirements

GEOS is python3 only. If you don't have python, I recommend downloading Anaconda Python.

1.2 Install GEOS

Usually, it's easiest to install *GEOS* through `pip`:

```
pip install geos
```

Alternatively, you can install *GEOS* from the github sources:

```
git clone git@github.com:grst/geos.git
cd geos
pip install -e geos
```

CHAPTER 2

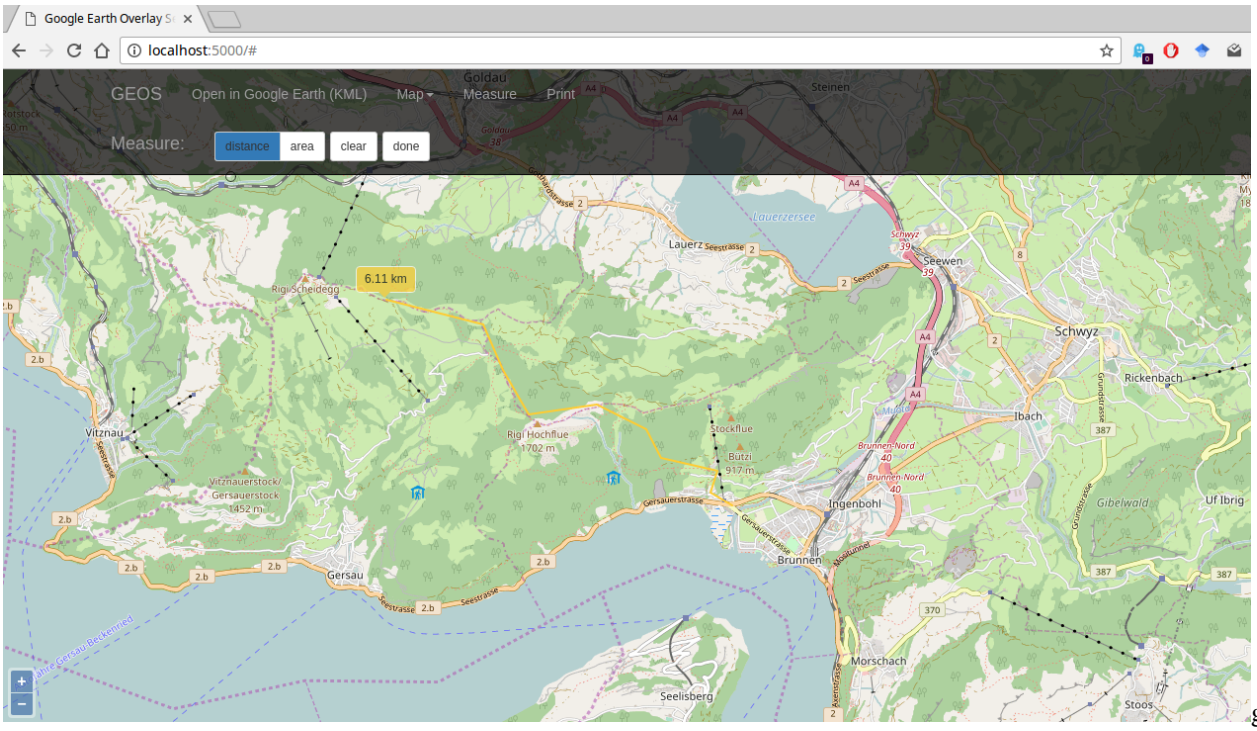
Usage

```
usage: geos [-h] [-m MAPSOURCE] [-H HOST] [-P PORT]
            [--display-host DISPLAY_HOST] [--display-port DISPLAY_PORT]
            [--display-scheme DISPLAY_SCHEME]

optional arguments:
  -h, --help            show this help message and exit
  -m MAPSOURCE, --mapsource MAPSOURCE
                        path to the directory containing the mapsource files.
                        [default: integrated mapsources]
  -H HOST, --host HOST  Hostname of the Flask app [default localhost]
  -P PORT, --port PORT  Port for the Flask app [default 5000]
  --display-host DISPLAY_HOST
                        Hostname used for self-referencing links [defaults to
                        Flask hostname]
  --display-port DISPLAY_PORT
                        Port used for self-referencing links [defaults to
                        Flask port]
  --display-scheme DISPLAY_SCHEME
                        URI-scheme used for self-referencing links [default
                        http]
```

To try out *GEOS*, simply open a terminal, type `geos` and hit enter! A web server will start. Note, that by default, the webserver is only reachable locally. You can adjust this using the `-H` parameter. If you use GEOS with a public url, e.g. `http://geos.example.com`, you can adjust the public hostname, port and scheme using the `--display-*` arguments.

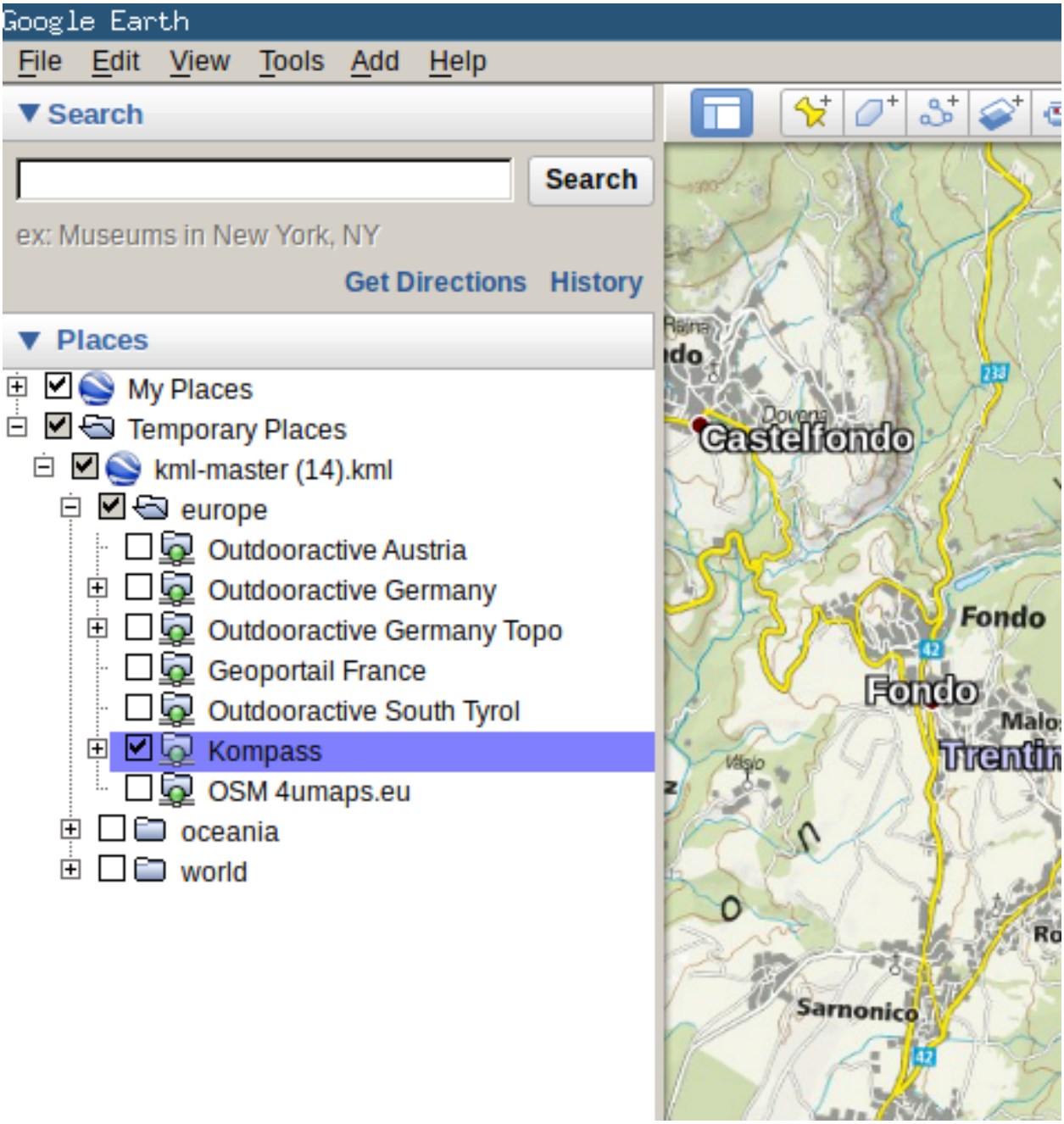
Open your browser and navigate to the URL. A web page will displaying a map and a menu bar. You can use the menu bar to choose between maps. Per default, it only contains the OSM Mapnik. From the menu bar, you can also choose tools to measure, draw and print maps.



geos-web

2.1 Open in Google Earth

Choose *"Open in Google Earth (KML)"* from the menu bar to download a KML file which you can open in Google Earth. The KML file will appear in the 'places' pane. Activate the checkbox of the map you want to display there:



Once the checkbox is activated, the mapoverlay should load. Note, that some maps do not provide tiles below a certain zoom level. In that case you have to zoom in for the tiles to load.

CHAPTER 3

More maps!

GEOS uses XML Mapsource files, to tell the server where it can find the tiles. I started a collection of such mapsources in a dedicated Github repository.

You can specify a directory containing xml mapsources using the `-m` command line parameter. *GEOS* will load all maps from that directory and put them into the kml file.

So, to start off, you can do the following:

```
git clone git@github.com:grst/mapsources.git
geos -m mapsources
```

Of course, you can create your own maps, too! If you do so, it would be cool if you shared them, e.g. by creating a pull request to the mapsources repo.

CHAPTER 4

Creating Mapsources

Essentially, the mapsources for *GEOS* are based on the MOBAC Mapsource XML Format.

A minimal Mapsource file for *GEOS* looks like this:

```
<customMapSource>
    <name>Example Map</name>  <!-- Name of the map as displayed in Google Earth -->
    <minZoom>5</minZoom>      <!-- minimal zoom level supported by the web map -->
    <maxZoom>15</maxZoom>     <!-- maximal zoom level supported by the web map -->
    <!-- url: tells GEOS where to find the tiles. Tile URLs contain three
    Parameters: zoom, x and y -->
    <url>http://example.com/map?zoom={$z}&amp;x={$x}&amp;y={$y}</url>
</customMapSource>
```

Additonally, *GEOS* currently supports the following optional parameters:

```
    <id>example_id</id>                    <!-- unique map identifier. If not␣
↪specified,
                                                the filename will be used as id -->
    <folder>europe/switzerland</folder>    <!-- use this tag to organize your maps in␣
↪Folders
                                                which will show up in Google Earth.␣
↪If not specified,
                                                GEOS will try to obtain the folder␣
↪from the directory
                                                tree, in which the mapsources are␣
↪saved in. -->
    <region>                               <!-- Set map boundaries. No tiles will␣
↪load outside -->
        <north>54.5</north>                <!-- Use geographic coordinates here.  -->
        <south>40</south>
        <east>15</east>
        <west>5</west>
   </region>
```

4.1 Multi Layer Mapsources

GEOS supports Mapsources which consist of multiple layers. Such a file looks as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<customMultiLayerMapSource>
   <name>Custom OSM Mapnik with Hills (Ger)</name>
   <layers>
      <customMapSource>
         <name>Custom OSM Mapnik</name>
         <minZoom>0</minZoom>
         <maxZoom>18</maxZoom>
         <url>http://tile.openstreetmap.org/{$z}/{$x}/{$y}.png</url>
      </customMapSource>
      <customMapSource>
         <name>cycling trails</name>
         <minZoom>0</minZoom>
         <maxZoom>18</maxZoom>
         <url>https://tile.waymarkedtrails.org/cycling/{$z}/{$x}/{$y}.png</url>
       </customMapSource>
   </layers>
</customMultiLayerMapSource>
```

4.2 Running GEOS in a docker container

If you are planning to run GEOS in a docker container, there is no requirement apart from having docker installed on your host. No install of python is necessary.

Running GEOS involves building a GEOS image, a one-time operation, and running a container when needed.

1. **Getting a Dockerfile**

When running GEOS in docker, you have 2 options:

- running the GEOS pip package release
- running GEOS built from source code

This choice will lead to using different Dockerfiles, as explained below.

Option a): using the pip release of GEOS

You do not need to download the GEOS sources. You only need to create a Dockerfile with the following contents:

```
FROM continuumio/miniconda3
RUN pip install geos
```

Option b): building GEOS from source code

The GEOS sources already contain the appropriate Dockerfile. Download the sources with

```
git clone https://github.com/grst/geos.git
```

1. **Building the GEOS docker image**

To build the docker image, move to the directory where the Dockerfile is and run :

```
docker build -t geos .
```

1. **Running the GEOS container**

Run (on a single line) :

```
docker run
--rm
-p <server_port>:5000
--mount type=bind,source=<server_mapsources_directory>,target=/opt/conda/lib/python3.
↪7/site-packages/geos/mapsources
geos
geos --host 0.0.0.0 --display-host <server_ip>
```

You will have to substitute the following variables with values that are relevant to your setup :

| Variable |Meaning |Example| |———————————|———————————————————|—-| |<server_port> |Port used to reach the server |5000| |<server_mapsources_directory>|Path to the `mapsources` directory on the server|/home/user/me/mapsources| |<server_ip> |IP adress of the server |192.168.0.1 / See note below|

Note:

- On Linux systems, <server_ip> can be found by running `ip route get 1 | awk '{print $NF; exit}'`

1. **Bonus: Building and running the GEOS container with docker-compose**

For ease of use, you can completely avoid using the docker *build* and *run* commands by creating a **docker-compose.yml** file next to the Dockerfile. Its contents can be similar to the following :

```
version: '3.7'
services:
    geos:
        build: .
        image: geos
        container_name: geos
        ports:
            - '5000:5000'
        volumes:
            - //c/Users/me/Documents/mapsources:/opt/conda/lib/python3.7/site-
↪packages/geos/mapsources
        command: geos --host 0.0.0.0
```

Note: The above file demonstrates the use of a Windows path for the *mapsources* host directory.

You can then start GEOS by issuing

```
docker-compose up
```

This command will take care of building the GEOS image if it does not exist locally.

You can stop GEOS by issuing

```
docker-compose down
```

CHAPTER 5

Using GEOS as library

5.1 Reading mapsources

class geos.mapsource.**MapLayer**(*tile_url=None*, *min_zoom=1*, *max_zoom=17*)
The layer object contained in a MapSource.

A MapSource can contain multiple layers. A layer contains all information on how to access the tiles.

Parameters

- **tile_url** – URL to the tiles which {$z}, {$x} and {$y} as placeholders.
- **min_zoom** (*int*) – minimal zoom level at which the layer is active
- **max_zoom** (*int*) – maximal zoom level at which the layer is active

get_tile_url(*zoom*, *x*, *y*)
Fill the placeholders of the tile url with zoom, x and y.

```
>>> ms = MapSource.from_xml("mapsources/osm.xml")
>>> ms.layers[0].get_tile_url(42, 43, 44)
'http://tile.openstreetmap.org/42/43/44.png'
```

get_tile_urls

class geos.mapsource.**MapSource**(*id*, *name*, *folder=''*, *bbox=None*)
A MapSource contains Meta-Information of the map. Additionally it can hold one or more MapLayers which contain the information on how to access the tiles.

static from_xml(*xml_path*, *mapsource_prefix=''*)
Create a MapSource object from a MOBAC mapsource xml.

Parameters

- **xml_path** – path to the MOBAC mapsource xml file.
- **mapsource_prefix** – root path of the mapsource folder. Used to determine relative path within the maps directory.

Note: The Meta-Information is read from the xml <id>, <folder>, <name> tags. If <id> is not available it defaults to the xml file basename. If <folder> is not available if defaults to the folder of the xml file with the *mapsource_prefix* removed.

The function first tries <url>, <minZoom>, <maxZoom> from <customMapSource> tags within the <layers> tag. If the <layers> tag is not available, the function tries to find <url>, <minZoom> and <maxZoom> on the top level. If none of thie information is found, a MapSourceException is raised.

Returns

Return type *MapSource*

Raises *MapSourceException* – when the xml file could not be parsed properly.

max_zoom
Get the maximal zoom level of all layers.

Returns the maximum of all zoom levels of all layers

Return type int

Raises ValueError – if no layers exist

min_zoom
Get the minimal zoom level of all layers.

Returns the minimum of all zoom levels of all layers

Return type int

Raises ValueError – if no layers exist

static parse_xml_boundary(*xml_region*)
Get the geographic bounds from an XML element

Parameters **xml_region** (*Element*) – The <region> tag as XML Element

Returns

Return type *GeographicBB*

static parse_xml_layer(*xml_custom_map_source*)
Get one MapLayer from an XML element

Parameters **xml_custom_map_source** (*Element*) – The <customMapSource> element tag wrapped in a <layers> tag as XML Element

Returns

Return type *MapLayer*

static parse_xml_layers(*xml_layers*)
Get the MapLayers from an XML element

Parameters **xml_layers** (*Element*) – The <layers> tag as XML Element

Returns

Return type list of MapLayer

exception geos.mapsource.**MapSourceException**

geos.mapsource.**load_maps**(*maps_dir*)
Load all xml map sources from a given directory.

Parameters **maps_dir** – path to directory to search for maps

Returns

Return type dict of MapSource

geos.mapsource.**walk_mapsources**(*mapsources*, *root=''*)
recursively walk through foldernames of mapsources.

Like os.walk, only for a list of mapsources.

Parameters **mapsources** (*list of MapSource*) –

Yields (root, foldernames, maps)

```
>>> mapsources = load_maps("test/mapsources")
>>> pprint([x for x in walk_mapsources(mapsources.values())])
[('',
  ['asia', 'europe'],
  [<MapSource: osm1 (root 1), n_layers: 1, min_zoom:5, max_zoom:18>,
   <MapSource: osm10 (root 2), n_layers: 1, min_zoom:5, max_zoom:18>]),
 ('/asia',
  [],
  [<MapSource: osm6 (asia), n_layers: 1, min_zoom:5, max_zoom:18>]),
 ('/europe',
  ['france', 'germany', 'switzerland'],
  [<MapSource: osm4 (eruope 1), n_layers: 1, min_zoom:1, max_zoom:18>]),
 ('/europe/france',
  [],
  [<MapSource: osm2 (europe/france 1), n_layers: 1, min_zoom:5, max_zoom:18>,
   <MapSource: osm3 (europe/france 2), n_layers: 1, min_zoom:1, max_zoom:18>,
   <MapSource: osm5 (europe/france 3), n_layers: 1, min_zoom:5, max_zoom:18>]),
 ('/europe/germany',
  [],
  [<MapSource: osm7 (europe/germany 1), n_layers: 1, min_zoom:5, max_zoom:18>,
   <MapSource: osm8 (europe/germany 2), n_layers: 1, min_zoom:5, max_zoom:18>]),
 ('/europe/switzerland',
  [],
  [<MapSource: osm9 (europe/switzerland), n_layers: 1, min_zoom:5, max_zoom:18>])]
```

5.2 Printing maps

Module for printing maps / exporting them as pdf

exception geos.print.**MapPrintError**

geos.print.**add_scales_bar**(*img*, *bbox*)
Add a scales bar to the map.

Calculates the resolution at the current latitude and inserts the corresponding scales bar on the map.

Parameters

- **img** (*Image*) – Image object to which the scales bar will be added.
- **bbox** (*TileBB*) – boundaries of the map

geos.print.**download_tile**(*map_layer*, *zoom*, *x*, *y*)
Download a given tile from the tile server.

Parameters

- **map_layer** (MapLayer) – MapLayer object which provides the tile-url.
- **zoom** (*int*) – zoom level
- **x** (*int*) – Tile-x-coordinate
- **y** (*int*) – Tile-y-coordinate

Returns temporary file containing the downloaded image.

Return type file

geos.print.**dpi_to_dpmm**(*dpi*)
Convert dpi (dots per inch) to dpmm (dots per millimeter)

Parameters **dpi** (*int*) –

Returns dots per millimeter

Return type float

geos.print.**get_print_bbox**(*x*, *y*, *zoom*, *width*, *height*, *dpi*)
Calculate the tile bounding box based on position, map size and resolution.

The function returns the next larger tile-box, that covers the specified page size in mm.

Parameters

- **x** (*float*) – map center x-coordinate in Mercator projection (EPSG:4326)
- **y** (*float*) – map center y-coordinate in Mercator projection (EPSG:4326)
- **zoom** (*int*) – tile zoom level to use for printing
- **width** (*float*) – page width in mm
- **height** (*float*) – page height in mm
- **dpi** (*int*) – resolution in dots per inch

Returns Bounding box of the map in TileCoordinates.

Return type *GridBB*

```
>>> str(get_print_bbox(4164462.1505763642, 985738.7965919945, 14, 297, 150, 120))
'<tile min: <zoom: 14, x: 9891, y: 7786>, max: <zoom: 14, x: 9897, y: 7790>>'
```

geos.print.**get_tiles**(*map_layer*, *bbox*, *n_workers=16*)
Download tiles.

Parameters

- **map_source** (MapSource) –
- **bbox** (*TileBB*) – Bounding box delimiting the map
- **n_workers** (*int*) – number of threads to used for downloading.

Returns

Dictionary mapping coordinates to temporary files.

Example:

```
{
    (x, y) : <FileHandle>
}
```

Return type dict of file

geos.print.**print_map**(*map_source*, *x*, *y*, *zoom=14*, *width=297*, *height=210*, *dpi=300*, *format='pdf'*)
Download map tiles and stitch them together in a single image, ready for printing.

Parameters

- **map_source** (MapSource) – Map to download
- **x** (*float*) – map center x-coordinate in Mercator projection (EPSG:4326)
- **y** (*float*) – map center y-coordinate in Mercator projection (EPSG:4326)
- **zoom** (*int*) – tile zoom level to use for printing
- **width** (*float*) – page width in mm
- **height** (*float*) – page height in mm
- **dpi** (*int*) – resolution in dots per inch
- **format** (*str*) – output format. Anything supported by Pillow.Image.save. E.g. "pdf", "jpeg", "png".

Returns path of temporary output file.

Return type str

geos.print.**stitch_map**(*tiles*, *width*, *height*, *bbox*, *dpi*)
Merge tiles together into one image.

Parameters

- **tiles** (*list of dict of file*) – tiles for each layer
- **width** (*float*) – page width in mm
- **height** (*height*) – page height in mm
- **dpi** (*dpi*) – resolution in dots per inch

Returns merged map.

Return type PIL.Image

5.3 Geometry

Conversions between WGS84, Cartesian, Mercator and TMS coordinates and bounding boxes, useful for converting geographic view ports as generated by Google Earth to TMS bounding boxes

class geos.geometry.**CartesianCoordinate**(*x*, *y*, *z*)
Represents a coordinate in a geocentric Cartesian coordinate system

length()

class geos.geometry.**GeographicBB**(*min_lon=None*, *min_lat=None*, *max_lon=None*, *max_lat=None*)
A bounding box defined by two geographic coordinates

center()

intersection(*other*)

intersects(*other*)

to_mercator()

class geos.geometry.**GeographicCoordinate**(*lon=None*, *lat=None*, *height=0.0*)
Represents a WGS84 Datum

Parameters

- **lon** – longitude in degrees
- **lat** – latitude in degrees
- **height** – height in meters above the surface of the earth spheroid

to_cartesian()

to_mercator()

class geos.geometry.**GridBB**(*zoom*, *min_x*, *min_y*, *max_x*, *max_y*)
A bounding box defined by two grid coordinates

intersection(*other*)

intersections(*other*)

is_inside(*tile*)

class geos.geometry.**GridCoordinate**(*zoom*, *x*, *y*)
Represents a position in a worldwide grid.

encode_quad_tree()

zoom_in()

Yields *GridCoordinate* – the four tiles of the next zoom level

class geos.geometry.**MercatorBB**(*_min=<class 'geos.geometry.MercatorCoordinate'>*, *_max=<class 'geos.geometry.MercatorCoordinate'>*)
A bounding box defined by two mercator coordinates

to_tile(*zoom*)
Converts EPSG:900913 to tile coordinates in given zoom level

class geos.geometry.**MercatorCoordinate**(*x=None*, *y=None*)
Represents a coordinate in Spherical Mercator EPSG:900913

to_geographic()

to_tile(*zoom*)

class geos.geometry.**RegionCoordinate**(*zoom*, *x*, *y*, *log_tiles_per_row=0*)
Represents a region containing multiple tiles in a worldwide grid of regions.

A region is a square containing multiple tiles, e.g.:

```
 -- --
| 1| 2|
 -- --
| 3| 4|
 -- --
```

A region must contain at least one tile and each row must have a power of two of tiles. The size of the region is specified with log2(tiles per row per region), e.g.

- log_tiles_per_row = 0 means 2**0 = 1 tile
- log_tiles_per_row = 2 means 2**2 = 4 tiles per row, thus 16 tiles per region.

Parameters

- **zoom** (*int*) – clear.
- **x** (*int*) – clear.
- **y** (*int*) – clear.
- **log_tiles_per_row** (*int*) – size of the region as log2(tiles per row per region).
- **to be at least 0** (*needs*) –

geographic_bounds()

get_tiles()
Get all TileCoordinates contained in the region

zoom_in()
Yields: GridCoordinate: the four tiles of the next zoom level

class geos.geometry.**TileCoordinate**(*zoom*, *x*, *y*)
Represents a coordinate in a worldwide tile grid. (aka the google-system)

geographic_bounds()

resolution()
Get the tile resolution at the current position.

The scale in WG84 depends on

- the zoom level (obviously)
- the latitude
- the tile size

References

- http://wiki.openstreetmap.org/wiki/Slippy_map_tilenames#Resolution_and_Scale
- http://gis.stackexchange.com/questions/7430/what-ratio-scales-do-google-maps-zoom-levels-correspond-to

Returns meters per pixel

Return type float

to_geographic()

to_mercator()

zoom_in()
Yields: GridCoordinate: the four tiles of the next zoom level

geos.geometry.**bboxiter**(*tile_bounds*, *tiles_per_row_per_region=1*)
Iterate through a grid of regions defined by a TileBB.

Parameters

- **tile_bounds** (GridBB) –
- **tiles_per_row_per_region** – Combine multiple tiles in one region. E.g. if set to two, four tiles will be combined in one region. See *kml* module description for more details. Leaving the default value '1' simply yields all tiles in the bounding box.

Note: If the number of regions would not be an integer due to specification of the *tiles_per_row_per_region* parameter, the boundaries will be rounded to the next smaller or next larger integer respectively.

Example: We have the following bounding box with size 2x2 and set tiles_per_row_per_region = 2, delimited by the coordinates (x, y):

```
(5,5)--- ---
    |       |
    |       |
    |       |
     --- ---(9,7)
```

Although this could be represented in one single region with two tiles per row, it will create four regions:

```
(2,2)--- ---          (5/2 = 2.5 -> 2, 5/2 = 2.5 -> 2)
    |   |   |
     --- ---
    |   |   |
     --- ---(5,4)     (9/2 = 4.5 -> 5, 7/2 = 3.5 -> 4)
```

Yields *Tuple* – all tuples (x, y) in the region delimited by the TileBB

geos.geometry.**griditer**(*x*, *y*, *ncol*, *nrow=None*, *step=1*)

Iterate through a grid of tiles.

Parameters

- **x** (*int*) – x start-coordinate
- **y** (*int*) – y start-coordinate
- **ncol** (*int*) – number of tile columns
- **nrow** (*int*) – number of tile rows. If not specified, this defaults to ncol, s.t. a quadratic region is generated
- **step** (*int*) – clear. Analogous to range().

Yields *Tuple* –

all tuples (x, y) in the region delimited by (x, y), (x + ncol, y + ncol).

geos.geometry.**init_geometry**(*tilesize=256.0*)

Python Module Index

g

Index

R

S

T

W

Z