geotiler

Release 0.15.1

unknown

Jul 19, 2024

CONTENTS

Download | Bugs | Source Code

GeoTiler is a library to create maps using tiles from a map provider.

Note: Version 0.20.0 of GeoTiler will change API. Please use:

```
geotiler>=0.14.2,<0.20.0
```

as your dependency requirement if you need old API.

The main goal of the library is to enable a programmer to create maps using tiles downloaded from OpenStreetMap, Stamen or other map provider.

The maps can be used by interactive applications or to create data analysis graphs.

The code is based on Python port of Modest Maps project, which is licensed under BSD license. All subsequent changes are licensed under GPL v3 license.

CHAPTER

ONE

INTRODUCTION

1.1 Features

GeoTiler features are

1. Very easy map API, which can be used in interactive and non-interactive way with other libraries, i.e. matplotlib, Cairo, Qt, etc.
2. Supported multiple map tiles providers, i.e. OpenStreetMap, Stamen and Blue Marble.
3. Allow to configure map providers API keys.
4. Asynchronous and synchronous map tiles downloader.
5. Map tiles caching with Redis and support for custom caching strategies.
6. The library design supports extensibility. Implement custom map tiles providers, tiles downloading or caching strategies within minutes.

1.2 Installation

To install GeoTiler use pip:

```
pip install --user geotiler
```

Requirements

- Pillow library
- cytoolz library
- aiohttp library
- Python 3.8 or later

1.3 Project Status

The source code of GeoTiler is based on Python port of Modest Maps project.

The GeoTiler project initial focus has been on

- library API improvements
- simplifying library design
- user documentation
- ensuring OpenStreetMap, Stamen and Blue Marble map providers work
- reimplementing default map tiles caching to use LRU cache (and now we switched to no cache by default)
- implementing Redis based map tiles cache
- moving docstrings tests to unit test modules
- Python 3 support
- making code PEP-8 compliant

You can help by

- testing map providers and reporting bugs
- providing more examples, i.e. how to integrate GeoTiler with various GUI toolkits
- implementing new caching strategies, i.e. memcached or PyLRU
- working on unit tests
- making code even more PEP-8 compliant

CHAPTER

TWO

USING GEOTILER LIBRARY

2.1 Rendering Map

Rendering a map with GeoTiler library consists of two steps

- create map object
- render map as an image

Map object is created using *geotiler.Map* class. Its constructor requires any of the following combinations of map parameters

- center, zoom and size
- extent and zoom
- extent and size

For example, to create map using first combination of the parameters above:

```
>>> import geotiler
>>> map = geotiler.Map(center=(-6.069, 53.390), zoom=16, size=(512, 512))
>>> map.extent
(-6.074495315551752, 53.38671986409586, -6.063508987426756, 53.39327172612266)
```

After creating a map object, map can be controlled with *extent*, *center*, *zoom* and *size* attributes. Each attribute can influence other, i.e. changing *extent* will change map size. Refer to the *geotiler.Map* class documentation for details.

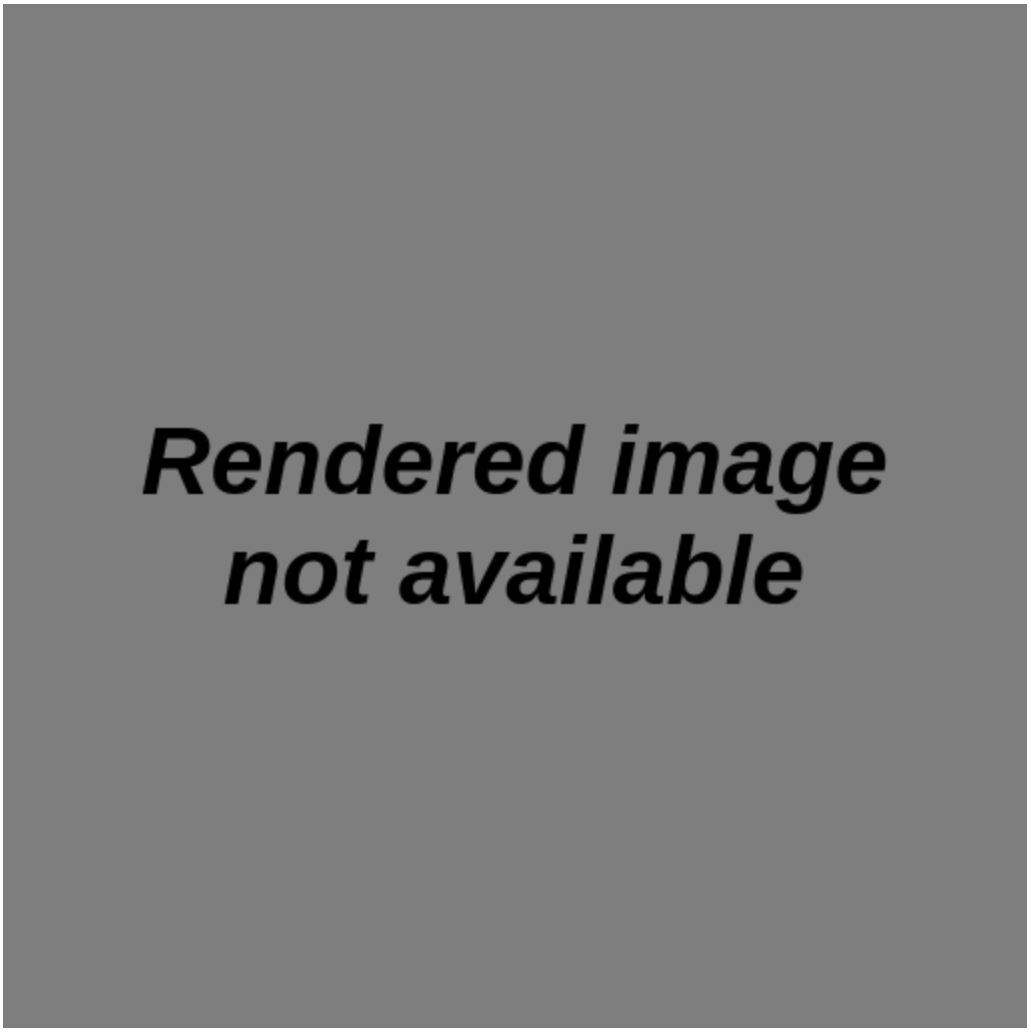
Having map object, the map tiles can be downloaded and rendered as an image with *geotiler.render_map()* function:

```
>>> image = geotiler.render_map(map)
```

The rendered image is an instance of `PIL.Image` class. Map image can be used with other library like matplotlib or Cairo to render additional map information, i.e. points of interests, GPS positions, text, etc. See *3rd Party Libraries* section for examples.

Alternatively, the map image can be simply saved as a file:

```
>>> image.save('map.png')
```


Rendered image
not available

2.2 Asynchronous Map Rendering

The *asyncio* Python framework enables programmers to write asynchronous, concurrent programs using coroutines.

GeoTiler allows to asynchronously download map tiles and render map image with *geotiler.render_map_async() asyncio* coroutine.

Very simple example to download a map using *asyncio* framework:

```
>>> coro = geotiler.render_map_async(map)
>>> loop = asyncio.get_event_loop()
>>> image = loop.run_until_complete(coro)
```

We include more complex example below. It reads location data from gpsd daemon, renders the map at the centre of current position and saves map to a file. There are two concurrent tasks running concurrently

- location reading
- map tiles downloading and rendering

The tasks communication is done via a queue holding current position.

```
async def read_gps(queue):
    """
    Read location data from `gpsd` daemon.
    """
    reader, writer = await asyncio.open_connection(port=2947)
    writer.write(b'?WATCH={"enable":true,"json":true}\n')
    while True:
        line = await reader.readline()
        data = json.loads(line.decode())
        if 'lon' in data:
            await queue.put((data['lon'], data['lat']))

async def show_map(queue, map):
    """
    Save map centered at location to a file.
    """
    while True:
        pos = yield from queue.get()

        map.center = pos
        img = await render_map_async(map)
        img.save('ex-async-gps.png', 'png')


size = 800, 800
start = 0, 0
mm = geotiler.Map(size=size, center=start, zoom=16)

queue = asyncio.Queue(1)        # queue holding current position from gpsd

# run location and map rendering tasks concurrently
t1 = show_map(queue, mm)
t2 = read_gps(queue)
```

(continues on next page)

(continued from previous page)

```
task = asyncio.gather(t1, t2)
loop = asyncio.get_event_loop()
loop.run_until_complete(task)
```

2.3 Map Providers

GeoTiler supports multiple map providers.

The list of supported map providers is presented in the table below. The default map provider is OpenStreetMap.

Provider	Provider Id	API Key Reference	License
OpenStreetMap	osm		Open Data Commons Open Database License
Stamen Toner	stamen-toner	stadiamaps	Creative Commons Attribution (CC BY 3.0) license
Stamen Toner Lite	stamen-toner-lite	stadiamaps	Stadia Maps Required Attribution
Stamen Terrain	stamen-terrain	stadiamaps	Stadia Maps Required Attribution
Stamen Terrain Background	stamen-terrain-background	stadiamaps	Stadia Maps Required Attribution
Stamen Terrain Lines	stamen-terrain-lines	stadiamaps	Stadia Maps Required Attribution
Stamen Water Color	stamen-watercolor	stadiamaps	Stadia Maps Required Attribution
Modest Maps Blue Marble	bluemarble		NASA guideline
OpenCycleMap	thunderforest-cycle	thunderforest	Thunderforest Terms and Conditions

A map provider might require API key. To add an API key for a map provider, the *$HOME/.config/geotiler/geotiler.ini* file has to be created with API key reference pointing to an API key, for example:

```
[api-key]
thunderforest = <api-key>
```

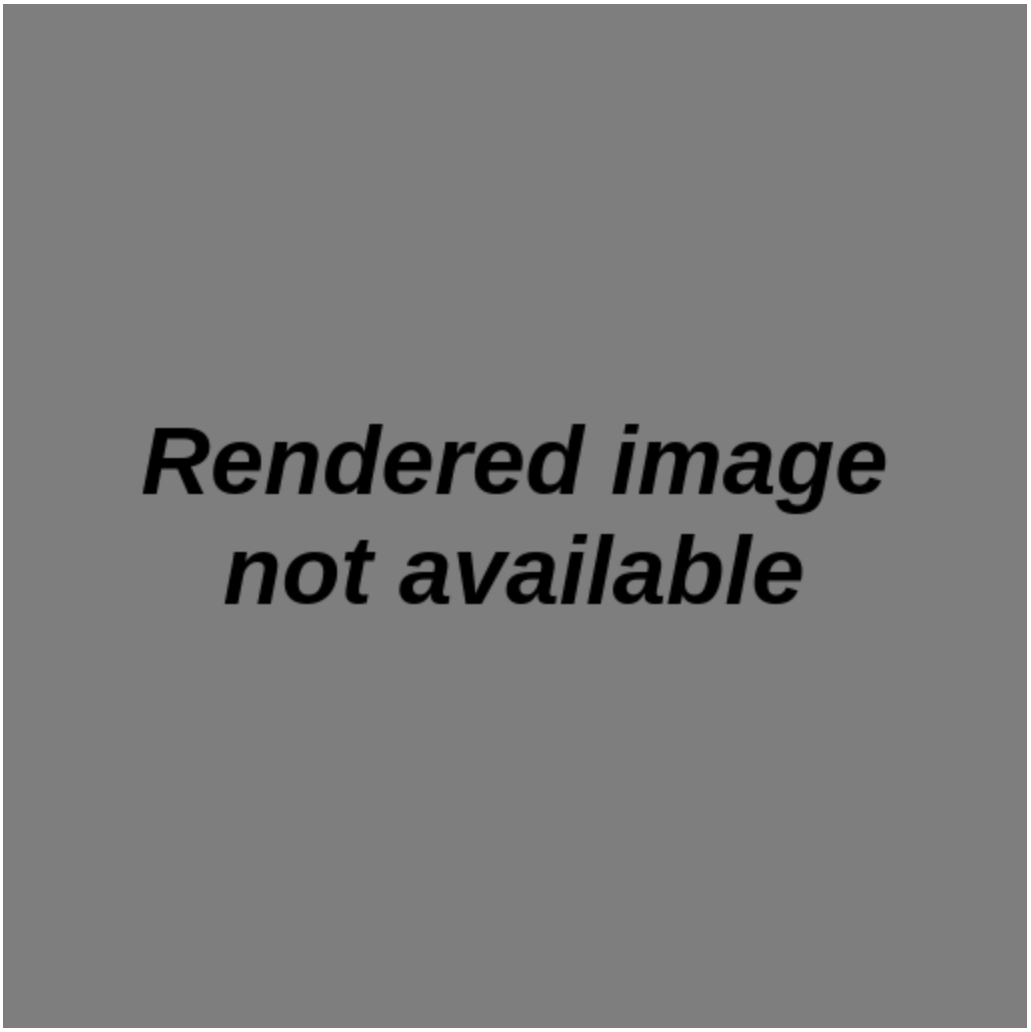
where <*api-key*> is usually a fixed size, alphanumeric hash value provided by map provider service.

Identificators of GeoTiler map providers can be listed with *geotiler.providers()* function. Map provider identificator can be used with *geotiler.find_provider()* function to create instance of map provider or it can be passed to *geotiler.Map* class constructor:

```
>>> map = geotiler.Map(center=(-6.069, 53.390), zoom=16, size=(512, 512), provider=
↪'stamen-toner')
>>> image = geotiler.render_map(map)

# or

>>> map = geotiler.Map(center=(-6.069, 53.390), zoom=16, size=(512, 512))
>>> map.provider = geotiler.find_provider('stamen-toner')
>>> image = geotiler.render_map(map)
```


Rendered image
not available

2.4 3rd Party Libraries

Map image rendered by GeoTiler can be used with other library like matplotlib or Cairo to draw additional map information or use the map in data analysis graphs.

GeoTiler implements various examples of such integration. The examples are presented in the subsections below.

2.4.1 Cairo Example

```
import cairo
import logging
import math

logging.basicConfig(level=logging.DEBUG)

import geotiler

bbox = 11.78560, 46.48083, 11.79067, 46.48283

#
# download background map using default map tiles provider - OpenStreetMap
#
mm = geotiler.Map(extent=bbox, zoom=18)
width, height = mm.size

img = geotiler.render_map(mm)

#
# create cairo surface
#
buff = bytearray(img.convert('RGBA').tobytes('raw', 'BGRA'))
surface = cairo.ImageSurface.create_for_data(
    buff, cairo.FORMAT_ARGB32, width, height
)
cr = cairo.Context(surface)

#
# plot circles around custom points
#
x0, y0 = 11.78816, 46.48114 # http://www.openstreetmap.org/search?query=46.48114%2C11.
↪78816
x1, y1 = 11.78771, 46.48165 # http://www.openstreetmap.org/search?query=46.48165%2C11.
↪78771
points = ((x0, y0), (x1, y1))
points = (mm.rev_geocode(p) for p in points)
for x, y in points:
    cr.set_source_rgba(0.0, 0.0, 1.0, 0.1)
    cr.arc(x, y, 30, 0, 2 * math.pi)
    cr.fill()

surface.write_to_png('ex-cairo.png')
```

2.4.2 matplotlib Example

```
import matplotlib.pyplot as plt

import logging
logging.basicConfig(level=logging.DEBUG)

import geotiler

bbox = 11.78560, 46.48083, 11.79067, 46.48283

fig = plt.figure(figsize=(10, 10))
ax = plt.subplot(111)

#
# download background map using OpenStreetMap
#
mm = geotiler.Map(extent=bbox, zoom=18)

img = geotiler.render_map(mm)
ax.imshow(img)

#
# plot custom points
#
x0, y0 = 11.78816, 46.48114 # http://www.openstreetmap.org/search?query=46.48114%2C11.
↪78816
x1, y1 = 11.78771, 46.48165 # http://www.openstreetmap.org/search?query=46.48165%2C11.
↪78771
points = ((x0, y0), (x1, y1))
x, y = zip(*(mm.rev_geocode(p) for p in points))
ax.scatter(x, y, c='red', edgecolor='none', s=10, alpha=0.9)

plt.savefig('ex-matplotlib.pdf', bbox_inches='tight')
plt.close()
```

2.4.3 Matplotlib Basemap Toolkit Example

```
import matplotlib.pyplot as plt
from mpl_toolkits.basemap import Basemap

import logging
logging.basicConfig(level=logging.DEBUG)

import geotiler

bbox = 11.78560, 46.48083, 11.79067, 46.48283

fig = plt.figure(figsize=(10, 10))
ax = plt.subplot(111)

```

(continues on next page)

(continued from previous page)

```
#
# download background map using OpenStreetMap
#
mm = geotiler.Map(extent=bbox, zoom=18)

img = geotiler.render_map(mm)

#
# create basemap
#
map = Basemap(
    llcrnrlon=bbox[0], llcrnrlat=bbox[1],
    urcrnrlon=bbox[2], urcrnrlat=bbox[3],
    projection='merc', ax=ax
)

map.imshow(img, interpolation='lanczos', origin='upper')

#
# plot custom points
#
x0, y0 = 11.78816, 46.48114 # http://www.openstreetmap.org/search?query=46.48114%2C11.
↪78816
x1, y1 = 11.78771, 46.48165 # http://www.openstreetmap.org/search?query=46.48165%2C11.
↪78771
x, y = map((x0, x1), (y0, y1))
ax.scatter(x, y, c='red', edgecolor='none', s=10, alpha=0.9)

plt.savefig('ex-basemap.pdf', bbox_inches='tight')
plt.close()
```

2.5 Caching

GeoTiler allows to cache map tiles. The *geotiler.cache.caching_downloader()* function enables us to adapt any caching strategy.

Beside generic caching downloader adapter, GeoTiler provides Redis store adapter. While it requires Redis server and Python Redis module installed, such solution gives map tiles persistence and advanced cache management.

The Redis cache example illustrates how Redis can be user for map tiles caching.

```
import functools
import redis

import geotiler
from geotiler.cache import redis_downloader

import logging
logging.basicConfig(level=logging.DEBUG)

# create tile downloader with Redis client as cache
```

(continues on next page)

(continued from previous page)

```
client = redis.Redis('localhost')
downloader = redis_downloader(client)

# use map renderer with new downloader
render_map = functools.partial(geotiler.render_map, downloader=downloader)

bbox = 11.78560, 46.48083, 11.79067, 46.48283
mm = geotiler.Map(extent=bbox, zoom=18)

# render the map for the first time...
img = render_map(mm)

# ... and second time to demonstrate use of the cache
img = render_map(mm)

# show some recent keys
print('recent cache keys {}'.format(client.keys()[:10]))
```

CHAPTER

THREE

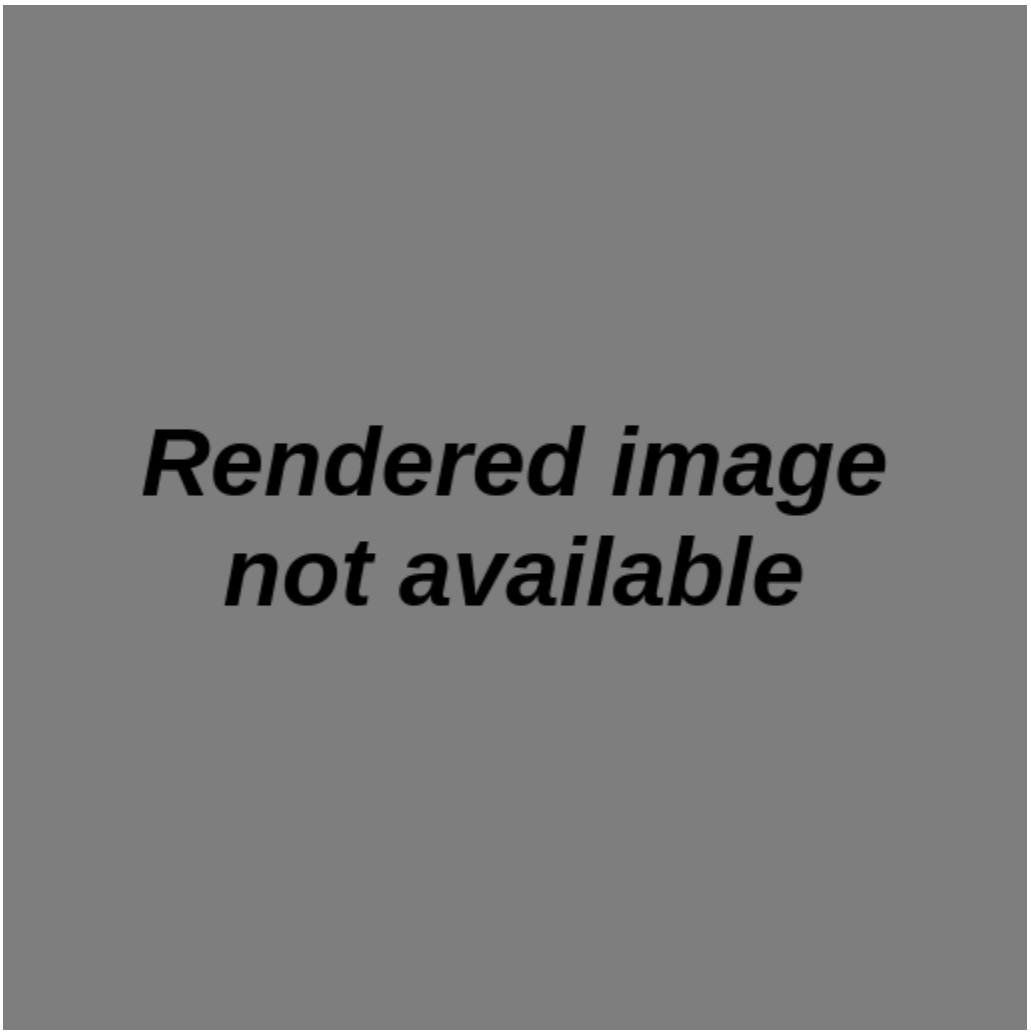
COMMAND LINE TOOLS

3.1 GeoTiler Lint Script

GeoTiler provides *geotiler-lint* script, which can be used to create a map image from command line.

For example, to create a map image using Blue Marble map tiles provider, map center, zoom and map image size:

```
geotiler-lint -c -6.069 53.390 -z 8 -s 512 512 -p bluemarble map-bluemarble.png
```



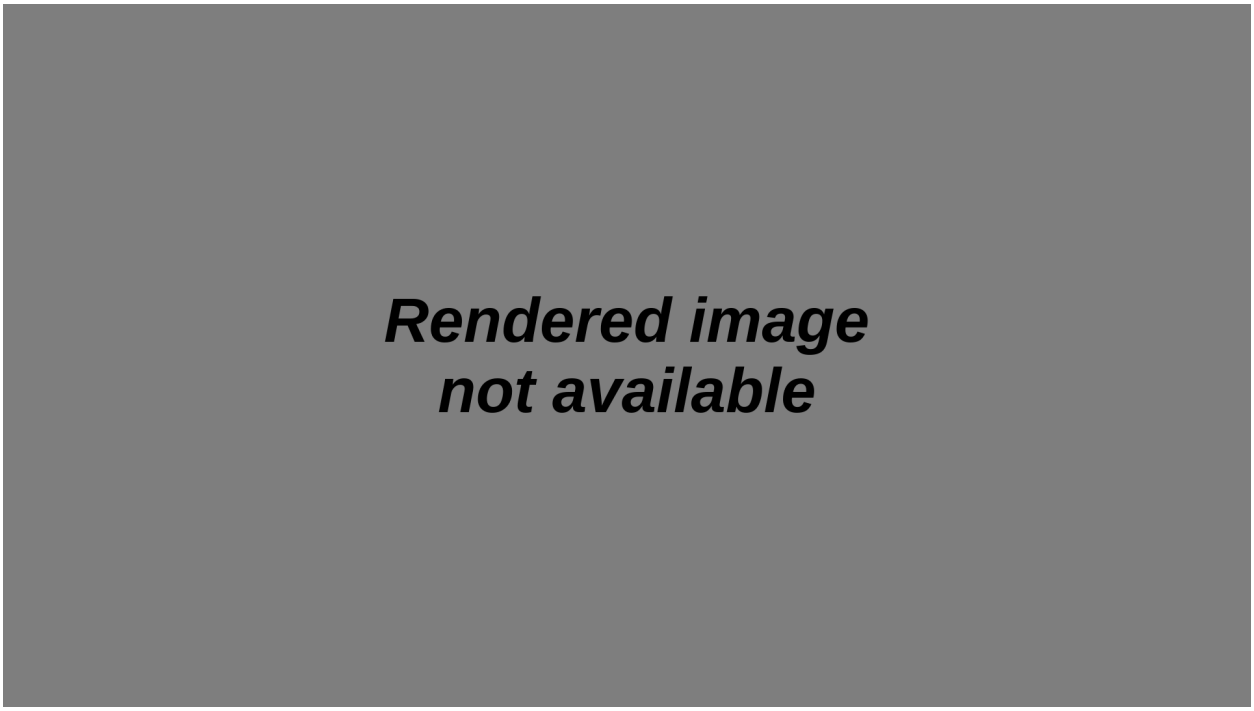
The script supports all implemented map tiles providers. Map can be specified with any of the required map parameters combination. It allows to switch map tiles caching strategy to use Redis cache.

3.2 Route Drawing

The *geotiler-route* script can be used to draw position information on a map. The script uses GPX file (can be compressed with *Gzip*, *bzip2* or *xz*) as its input.

For example:

```
geotiler-route path.gpx map-path.png
```



The map extents are determined by aggregating positions stored in the input file. The map zoom is automatically calculated from the map extent and map image size.

3.3 Map Tiles Fetching

The *geotiler-fetch* script enables us to fetch map tiles and store them in cache to be reused later by a map application. Optionally, map image can be saved to a file for each zoom level.

To fetch OpenCycleMap tiles and save map images into files named *map-01.png', `map-02.png*, ..., *map-19.png*:

```
geotiler-fetch -p tunderforest-cycle -f 'map-{:02d}.png' -6.0759 53.3830 -6.0584 53.3945
```

CHAPTER

FOUR

API

4.1 Map Rendering

geotiler.Map([extent, center, zoom, size, ...])	Map created from tiles and to be drawn as an image.
geotiler.render_map(map[, tiles, downloader])	Download map tiles and render map image.
geotiler.render_map_async(map[, tiles, ...])	Asyncio coroutine to download map tiles asynchronously and render map image.
geotiler.fetch_tiles(map[, downloader])	Create and fetch map tiles.
geotiler.providers()	Get sorted list of all map providers identificators.
geotiler.find_provider(id)	Load map provider data from JSON file and create map provider.

class geotiler.**Map**(*extent=None*, *center=None*, *zoom=None*, *size=None*, *provider: str | MapProvider = 'osm'*)

Map created from tiles and to be drawn as an image.

The extent, zoom, center and image size can be changed at any time with appropriate properties.

Variables

- **provider** – Map tiles provider identificator (default *osm*) or object.
- **_zoom** – Map zoom attribute accessed via *zoom* property.
- **_size** – Map image size accessed via *size* property.
- **origin** – Tile coordinates at map zoom level of base tile.
- **offset** – Position of base tile relative to map center.

__init__(*extent=None*, *center=None*, *zoom=None*, *size=None*, *provider: str | MapProvider = 'osm'*) → None

Create map.

One of the following parameters combination is required

- center, zoom and size
- extent and size
- extent and zoom

If none of above parameters combination is provided, then *ValueError* exception is raised.

Parameters

- **extent** – Map geographical extent.

- **center** – Map geographical center.
- **zoom** – Map zoom.
- **size** – Map image size.
- **provider** – Map tiles provider.

__str__()

Return str(self).

__weakref__

list of weak references to the object

property center

Calculate map geographical center.

It is a tuple of two values - longitude and latitude.

Setting map geographical center affects map geographical extent.

property extent

Calculate map geographical extent.

It is a tuple of 2 coordinates (longitude and latitude)

- lower-bottom corner of the map
- right-top corner of the map

Setting map extent changes map image size.

geocode(*point*)

Geocode map image point.

The method calculates geographical location (longitude, latitude) of image point.

Parameters
point – Image map point (x, y).

rev_geocode(*location*)

Reverse geocode geographical location.

The method calculates location position (x, y) on map image.

Parameters
location – Geographical location (longitude, latitude).

property size

Size of the image containing map.

It is a sequence of two integer values - width and height of the image.

Setting size of the image affects map geographical extent.

property zoom

Map zoom value.

Setting map value does *not* affect any other map properties like extent, center or image size.

geotiler.**render_map**(*map*, *tiles=None*, *downloader=None*, ***kw*)

Download map tiles and render map image.

If tiles are specified, then the provided tiles are used to render map. Otherwise, map tiles are downloaded from map provider.

If *downloader* is null, then default map tiles downloader is used (*geotiler.tile.io.fetch_tiles()*).

The function returns an image (instance of *PIL.Image* class).

Parameters

- **map** – Map instance.
- **tiles** – Optional map tiles.
- **downloader** – Map tiles downloader.
- **kw** – Parameters passed to the downloader.

async geotiler.**render_map_async**(*map*, *tiles=None*, *downloader=None*, ***kw*)

Asyncio coroutine to download map tiles asynchronously and render map image.

If tiles are specified, then the provided tiles are used to render map. Otherwise, map tiles are downloaded from map provider.

If *downloader* is null, then default map tiles downloader is used (*geotiler.tile.io.fetch_tiles()*).

The function returns an image (instance of *PIL.Image* class).

Parameters

- **map** – Map instance.
- **tiles** – Optional map tiles.
- **downloader** – Map tiles downloader.
- **kw** – Parameters passed to the downloader.

geotiler.**fetch_tiles**(*map*, *downloader=None*, ***kw*)

Create and fetch map tiles.

Asynchronous generator of map tiles is returned.

Parameters

- **map** – Map instance.
- **downloader** – Map tiles downloader.
- **kw** – Parameters passed to the downloader.

geotiler.**providers**()

Get sorted list of all map providers identificators.

geotiler.**find_provider**(*id*)

Load map provider data from JSON file and create map provider.

Parameters

id – Map provider identificator.

4.2 Tile Downloading and Caching

geotiler.cache.caching_downloader(get, set, ...)	Download tiles from cache and missing tiles with the downloader.
geotiler.cache.redis_downloader(client[, ...])	Create downloader using Redis as cache for map tiles.
geotiler.tile.io.fetch_tiles(tiles, num_workers)	Download map tiles.

async geotiler.cache.**caching_downloader**(*get*, *set*, *downloader*, *tiles*, *num_workers*, ***kw*)

Download tiles from cache and missing tiles with the downloader.

Asynchronous generator of map tiles is returned.

The code flow is

- caching downloader gets tile data from cache using URLs
- the original downloader is used to download missing tile data
- cache is updated with all existing tile data

The cache getter function (*get* parameter) should return *None* if tile data is not in cache for given URL.

A collection of tiles is returned.

Parameters

- **get** – Function to get a tile data from cache.
- **set** – Function to put a tile data in cache.
- **downloader** – Original tiles downloader (asyncio coroutine).
- **tiles** – Collection tiles to fetch.
- **num_workers** – Number of workers used to connect to a map provider service.
- **kw** – Parameters passed to downloader coroutine.

geotiler.cache.**redis_downloader**(*client*, *downloader=None*, *timeout=604800*)

Create downloader using Redis as cache for map tiles.

Parameters

- **client** – Redis client object.
- **downloader** – Map tiles downloader, use *None* for default downloader.
- **timeout** – Map tile data expiry timeout, default 1 week.

async geotiler.tile.io.**fetch_tiles**(*tiles*, *num_workers*)

Download map tiles.

Asynchronous generator of map tiles is returned.

A collection of tiles is returned. Each successfully downloaded tile has *Tile.img* attribute set. If there was an error while downloading a tile, then *Tile.img* is set to null and *Tile.error* to a value error.

Parameters

- **tiles** – Collection of tiles.
- **num_workers** – Number of workers used to connect to a map provider service.

CHAPTER

FIVE

CHANGELOG

5.1 0.15.1

- move stamen map tiles definitions to stadia maps

5.2 0.15.0

- update to new pillow library api
- python 3.9 or later is required

5.3 0.14.7

- do not use *a, b, c* subdomains for osm map provider as osm does not support them anymore

5.4 0.14.6

- add missing *pkg_resources* dependency to the build system metadata

5.5 0.14.5

- allow to pass map provider id or map provider data to a map constructor
- add tile width and height to the map provider data
- Python 3.8 is required

5.6 0.14.4

- raise error if a map provider requires API key and configuration file does not exist

5.7 0.14.3

- show provider map name when requesting its string representation; this also makes nicer string representation of map object

5.8 0.14.2

- python 3.7 is required since now
- keep center and extent of map unchanged when reseting zoom value to the same value
- remove some of the rounding of calculations to minimize calculation error

5.9 0.14.1

- fixed calculation of number of required tiles to render a map; this also improves performance of map rendering as less tiles is required

5.10 0.14.0

- corrected use of Redis API in Redis cache

5.11 0.13.0

- implemented function *geotiler.fetch_tiles* to allow processing of tiles before map rendering
- use map provider download limit to honour map provider service connection limits
- the `aiohttp` module is used again (with support for proxies)
- Python 3.6 is required

5.12 0.12.0

- there is no OSM cycle map anymore, just OpenCycleMap by Thunderforest; therefore the osm-cycle map provider is replaced with thunderforest-cycle one
- added support for map provider api keys

5.13 0.11.0

- added support for stamen-terrain-background and stamen-terrain-lines map providers

5.14 0.10.0

- added type check for map image size, which is a sequence of two integer values
- read map configuration files using Unicode encoding

5.15 0.9.0

- replace Python based map providers configuration with JSON file configuration (inspired by *poor-maps* project tile source definition https://github.com/otsaloma/poor-maps/tree/master/tilesources)
- *geotiler-lint* map provider argument is optional now
- *geotiler.Map* class constructor parameter for provider is changed to be string instead of map provider instance
- added caching example using *shelve* Python module (thanks to *matthijs876*)

5.16 0.8.0

- implemented *geotiler-fetch* script to cache map tiles upfront
- implemented *geotiler-route* script to draw GPX track on a map
- OpenCV example added, thanks to *matthijs876*

5.17 0.7.0

- use `urllib` Python standard library to fetch map tiles, which allows us to support proxies; `aiohttp` is no longer used
- implemented ChartBundle map providers

5.18 0.6.0

- map rendering performance improved
- Qt application example simplified and its performance improved

5.19 0.5.0

- format of rendered map image is changed from RGB to RGBA
- implemented example Qt application to show map centered at current position read from GPS; it uses asyncio and *quamash* library

5.20 0.4.0

- caching API has changed
- use asyncio to download map tiles
- implemented `geotiler.render_map_async()` function to download map asynchronously
- default LRU caching is gone

5.21 0.3.0

- implemented stamen-toner-lite map provider
- fixed stamen terrain and watercolor providers, which were broken since stamen started to use tiles in jpeg format

5.22 0.2.0

- improved error handling of map tiles downloading
- fixed geotiler-lint script installation issue
- documentation for geotiler-lint script added
- documented map tiles licensing information

5.23 0.1.0

- initial release
- genindex
- search

INDEX

Symbols

C

E

F

G

M

P

R

S

Z