
TileServer GL Documentation

Release 1.0

Klokan Technologies GmbH

Dec 19, 2018

1	Installation	3
1.1	Docker	3
1.2	npm	3
1.3	tileserver-gl-light on npm	4
1.4	From source	4
1.5	On OSX	4
2	Usage	5
2.1	Getting started	5
2.2	Default styles and configuration	5
3	Configuration file	7
3.1	options	8
3.2	styles	9
3.3	data	9
3.4	Referencing local files from style JSON	9
4	Deployment	11
4.1	Caching	11
4.2	Securing	11
4.3	Running behind a proxy or a load-balancer	11
5	Available endpoints	13
5.1	Styles	13
5.2	Rendered tiles	13
5.3	WMTS Capabilities	13
5.4	Static images	14
5.5	Source data	14
5.6	TileJSON arrays	14
5.7	List of available fonts	14
5.8	Health check	15
6	Indices and tables	17

Contents:

1.1 Docker

When running docker image, no special installation is needed – the docker will automatically download the image if not present.

Just run `docker run --rm -it -v $(pwd):/data -p 8080:80 klokantech/tileserver-gl`.

Additional options (see *Usage*) can be passed to the TileServer GL by appending them to the end of this command. You can, for example, do the following:

- `docker run ... klokantech/tileserver-gl my-tiles.mbtiles` – explicitly specify which mbtiles to use (if you have more in the folder)
- `docker run ... klokantech/tileserver-gl --verbose` – to see the default config created automatically

1.2 npm

Just run `npm install -g tileserver-gl`.

1.2.1 Native dependencies

There are some native dependencies that you need to make sure are installed if you plan to run the TileServer GL natively without docker. The precise package names you need to install may differ on various platforms.

These are required on Debian 9:

- `build-essential`
- `libcairo2-dev`
- `libprotobuf-dev`

1.3 tileserver-gl-light on npm

Alternatively, you can use `tileserver-gl-light` package instead, which is pure javascript (does not have any native dependencies) and can run anywhere, but does not contain rasterization features.

1.4 From source

Make sure you have Node v6 (nvm install 6) and run:

```
npm install
node .
```

1.5 On OSX

Make sure to have dependencies of `canvas` package installed:

```
brew install pkg-config cairo libpng jpeg giflib
```


2.1 Getting started

```
Usage: main.js tileserver-gl [mbtiles] [options]
```

Options:

-h, --help	output usage information
--mbtiles <file>	MBTiles file (uses demo configuration); ignored if the configuration file is also specified
-c, --config <file>	Configuration file [config.json]
-b, --bind <address>	Bind address
-p, --port <port>	Port [8080]
-C --no-cors	Disable Cross-origin resource sharing headers
-V, --verbose	More verbose output
-s, --silent	Less verbose output
-v, --version	Version info

2.2 Default styles and configuration

- If no configuration file is specified, the default styles (compatible with openmaptiles) are used.
- If no mbtiles file is specified (and is not found in the current working directory), an extract is downloaded directly from <https://openmaptiles.org/>

CHAPTER 3

Configuration file

The configuration file defines the behavior of the application. It's a regular JSON file.

Example:

```
{
  "options": {
    "paths": {
      "root": "",
      "fonts": "fonts",
      "sprites": "sprites",
      "styles": "styles",
      "mbtiles": ""
    },
    "domains": [
      "localhost:8080",
      "127.0.0.1:8080"
    ],
    "formatQuality": {
      "jpeg": 80,
      "webp": 90
    },
    "maxScaleFactor": 3,
    "maxSize": 2048,
    "pbfAlias": "pbf",
    "serveAllFonts": false,
    "serveStaticMaps": true
  },
  "styles": {
    "basic": {
      "style": "basic.json",
      "tilejson": {
        "type": "overlay",
        "bounds": [8.44806, 47.32023, 8.62537, 47.43468]
      }
    }
  },
}
```

(continues on next page)

(continued from previous page)

```
"hybrid": {
  "style": "satellite-hybrid.json",
  "serve_rendered": false,
  "tilejson": {
    "format": "webp"
  }
},
"data": {
  "zurich-vector": {
    "mbtiles": "zurich.mbtiles"
  }
}
```

3.1 options

3.1.1 paths

Defines where to look for the different types of input data.

The value of `root` is used as prefix for all data types.

3.1.2 domains

You can use this to optionally specify on what domains the rendered tiles are accessible. This can be used for basic load-balancing or to bypass browser's limit for the number of connections per domain.

3.1.3 frontPage

Path to the html (relative to `root` path) to use as a front page.

Use `true` (or nothing) to serve the default TileServer GL front page with list of styles and data. Use `false` to disable the front page altogether (404).

3.1.4 formatQuality

Quality of the compression of individual image formats. [0-100]

3.1.5 maxScaleFactor

Maximum scale factor to allow in raster tile and static maps requests (e.g. @3x suffix). Also see `maxSize` below. Default value is 3, maximum 9.

3.1.6 maxSize

Maximum image side length to be allowed to be rendered (including scale factor). Be careful when changing this value since there are hardware limits that need to be considered. Default is 2048.

3.1.7 minRendererPoolSizes

Minimum amount of raster tile renderers per scale factor. The value is an array: the first element is the minimum amount of renderers for scale factor one, the second for scale factor two and so on. If the array has less elements than `maxScaleFactor`, then the last element is used for all remaining scale factors as well. Selecting renderer pool sizes is a trade-off between memory use and speed. A reasonable value will depend on your hardware and your amount of styles and scale factors. If you have plenty of memory, you'll want to set this equal to `maxRendererPoolSizes` to avoid increased latency due to renderer destruction and recreation. If you need to conserve memory, you'll want something lower than `maxRendererPoolSizes`, possibly allocating more renderers to scale factors that are more common. Default is `[8, 4, 2]`.

3.1.8 maxRendererPoolSizes

Maximum amount of raster tile renderers per scale factor. The value and considerations are similar to `minRendererPoolSizes` above. If you have plenty of memory, try setting these equal to or slightly above your processor count, e.g. if you have four processors, try a value of `[6]`. If you need to conserve memory, try lower values for scale factors that are less common. Default is `[16, 8, 4]`.

3.1.9 watermark

Optional string to be rendered into the raster tiles (and static maps) as watermark (bottom-left corner). Can be used for hard-coding attributions etc. (can also be specified per-style). Not used by default.

3.2 styles

Each item in this object defines one style (map). It can have the following options:

- `style` – name of the style json file [required]
- `serve_rendered` – whether to render the raster tiles for this style or not
- `serve_data` – whether to allow access to the original tiles, sprites and required glyphs
- `tilejson` – properties to add to the TileJSON created for the raster data
 - `format` and `bounds` can be especially useful

3.3 data

Each item specifies one data source which should be made accessible by the server. It has the following options:

- `mbtiles` – name of the mbtiles file [required]

The mbtiles file does not need to be specified here unless you explicitly want to serve the raw data.

3.4 Referencing local files from style JSON

You can link various data sources from the style JSON (for example even remote TileJSONs).

3.4.1 MBTiles

To specify that you want to use local mbtiles, use the following syntax: `mbtiles://switzerland.mbtiles`. The TileServer-GL will try to find the file `switzerland.mbtiles` in `root + mbtiles path`.

For example:

```
"sources": {
  "source1": {
    "url": "mbtiles://switzerland.mbtiles",
    "type": "vector"
  }
}
```

Alternatively, you can use `mbtiles://{zurich-vector}` to reference existing data object from the config. In this case, the server will look into the `config.json` to determine what mbtiles file to use. For the config above, this is equivalent to `mbtiles://zurich.mbtiles`.

3.4.2 Sprites

If your style requires any sprites, make sure the style JSON contains proper path in the `sprite` property.

It can be a local path (e.g. `my-style/sprite`) or remote `http(s)` location (e.g. `https://mycdn.com/my-style/sprite`). Several possible extensions are added to this path, so the following files should be present:

- `sprite.json`
- `sprite.png`
- `sprite@2x.json`
- `sprite@2x.png`

You can also use the following placeholders in the `sprite` path for easier use:

- `{style}` – gets replaced with the name of the style file (`xxx.json`)
- `{styleJsonFolder}` – gets replaced with the path to the style file

3.4.3 Fonts (glyphs)

Similarly to the sprites, the style JSON also needs to contain proper paths to the font glyphs (in the `glyphs` property) and can be both local and remote.

It should contain the following placeholders:

- `{fontstack}` – name of the font and variant
- `{range}` – range of the glyphs

For example `"glyphs": "{fontstack}/{range}.pbf"` will instruct TileServer-GL to look for the files such as `fonts/Open Sans/0-255.pbf` (fonts come from the `paths` property of the `config.json` example above).

Typically - you should use nginx/lighttpd/apache on the frontend - and the tileserver-gl server is hidden behind it in production deployment.

4.1 Caching

There is a plenty of options you can use to create proper caching infrastructure: Varnish, CloudFlare, ...

4.2 Securing

Nginx can be used to add protection via https, password, referrer, IP address restriction, access keys, etc.

4.3 Running behind a proxy or a load-balancer

If you need to run TileServer GL behind a proxy, make sure the proxy sends `X-Forwarded-*` headers to the server (most importantly `X-Forwarded-Host` and `X-Forwarded-Proto`) to ensures the URLs generated inside TileJSON etc. are using the desired domain and protocol.

Available endpoints

If you visit the server on the configured port (default 8080) you can see your maps appearing in the browser.

5.1 Styles

- Styles are served at `/styles/{id}/style.json` (+ array at `/styles.json`)
 - Sprites at `/styles/{id}/sprite[@2x].{format}`
 - Fonts at `/fonts/{fontstack}/{start}-{end}.pbf`

5.2 Rendered tiles

- Rendered tiles are served at `/styles/{id}/{z}/{x}/{y}[@2x].{format}`
 - The optional `@2x` (or `@3x`, `@4x`) part can be used to render HiDPI (retina) tiles
 - Available formats: `png`, `jpg` (`jpeg`), `webp`
 - TileJSON at `/styles/{id}.json`
- The rendered tiles are not available in the `tileserver-gl-light` version.

5.3 WMTS Capabilities

- WMTS Capabilities are served at `/styles/{id}/wmts.xml`

5.4 Static images

- Several endpoints:
 - `/styles/{id}/static/{lon},{lat},{zoom}[@{bearing}[,{pitch}]]/{width}x{height}[@2x].{format}` (center-based)
 - `/styles/{id}/static/{minx},{miny},{maxx},{maxy}/{width}x{height}[@2x].{format}` (area-based)
 - `/styles/{id}/static/auto/{width}x{height}[@2x].{format}` (autofit path – see below)
- All the static image endpoints additionally support following query parameters:
 - `path` - comma-separated lng, lat, pipe-separated pairs
 - * e.g. `5.9,45.8|5.9,47.8|10.5,47.8|10.5,45.8|5.9,45.8`
 - `latlng` - indicates the path coordinates are in lat, lng order rather than the usual lng, lat
 - `fill` - color to use as the fill (e.g. `red`, `rgba(255,255,255,0.5)`, `#0000ff`)
 - `stroke` - color of the path stroke
 - `width` - width of the stroke
 - `padding` - “percentange” padding for fitted endpoints (area-based and path autofit)
 - * value of `0.1` means “add 10% size to each side to make sure the area of interest is nicely visible”
- You can also use (experimental) `/styles/{id}/static/raw/...` endpoints with raw spherical mercator coordinates (EPSG:3857) instead of WGS84.
- The static images are not available in the `tileserver-gl-light` version.

5.5 Source data

- Source data are served at `/data/{mbtiles}/{z}/{x}/{y}.{format}`
 - Format depends on the source file (usually `png` or `pbfb`)
 - * `geojson` is also available (useful for inspecting the tiles) in case the original format is `pbfb`
 - TileJSON at `/data/{mbtiles}.json`

5.6 TileJSON arrays

Array of all TileJSONs is at `/index.json` (`/rendered.json`; `/data.json`)

5.7 List of available fonts

Array of names of the available fonts is at `/fonts.json`

5.8 Health check

Endpoint reporting health status is at `/health` and currently returns:

- 503 Starting - for a short period before everything is initialized
- 200 OK - when the server is running

CHAPTER 6

Indices and tables

- [genindex](#)
- [modindex](#)
- [search](#)