gdal2tiles Documentation

Release 0.1.9

Tehamalab

Sep 30, 2020

Contents:

CHAPTER 1

gdal2tiles

A python library for generating map tiles inspired by gdal2tiles.py from GDAL project.

1.1 Dependancies

- GDAL development header files, sometimes available as *libgdal-dev* or *libgdal-devel* packages.

1.2 Installation

To install gdal2tiles library you can use pip:

```
$ pip install gdal2tiles
```

1.3 Basic usage

```
import gdal2tiles

gdal2tiles.generate_tiles('/path/to/input_file', '/path/to/output_dir/')
```

You can also pass various keyword as optional keyword arguments to *generate_tiles()* function. For example

```
gdal2tiles.generate_tiles('input_file', 'output_dir/', nb_processes=2, zoom='7-9')
```

OR

```
options = {'zoom': (7, 9), 'resume': True}
gdal2tiles.generate_tiles('input_file', 'output_dir/', **options)
```

In general

```
gdal2tiles.generate_tiles(input_file, output_folder, **options)
```

Arguments: `input_file` *(str)*: Path to input file.

`output_folder` *(str)*: Path to output folder.

`options`: Tile generation options.

Options:

`profile` ***(str)*****: Tile cutting profile (mercator,geodetic,raster) - default** 'mercator' (Google Maps compatible)

`resampling` ***(str)*****: Resampling method (average,near,bilinear,cubic,cubicsp** line,lanczos,antialias) - default 'average'

`s_srs`: The spatial reference system used for the source input data

`zoom`: Zoom levels to render; format: '[int min, int max]', 'min-max' or 'int/str zoomlevel'.

`tile_size` *(int)*: Size of tiles to render - default 256

`resume` *(bool)*: Resume mode. Generate only missing files.

`srcnodata`: NODATA transparency value to assign to the input data

`tmscompatible` ***(bool)*****: When using the geodetic profile, specifies the base** resolution as 0.703125 or 2 tiles at zoom level 0.

`verbose` *(bool)*: Print status messages to stdout

`kml` ***(bool)*****: Generate KML for Google Earth - default for 'geodetic'** profile and 'raster' in EPSG:4326. For a dataset with different projection use with caution!

`url` *(str)*: URL address where the generated tiles are going to be published

`webviewer` ***(str)*****: Web viewer to generate (all,google,openlayers,none) -** default 'all'

`title` *(str)*: Title of the map

`copyright` *(str)*: Copyright for the map

`googlekey` (str): Google Maps API key from http://code.google.com/apis/maps/signup.html

`bingkey` *(str)*: Bing Maps API key from https://www.bingmapsportal.com/

`nb_processes` *(int)*: Number of processes to use for tiling.

CHAPTER 2

Installation

2.1 Stable release

To install gdal2tiles library, run this command in your terminal:

```
$ pip install gdal2tiles
```

This is the preferred method to install gdal2tiles, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

2.2 From sources

The sources for gdal2tiles can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/tehamalab/gdal2tiles
```

Or download the tarball:

```
$ curl  -OL https://github.com/tehamalab/gdal2tiles/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER 3

Modules

3.1 gdal2tiles package

3.1.1 Submodules

3.1.2 gdal2tiles.gdal2tiles module

class gdal2tiles.gdal2tiles.**GDAL2Tiles**(*input_file*, *output_folder*, *options*)

Bases: object

generate_base_tiles()

Generation of the base tiles (the lowest in the pyramid) directly from the input raster

generate_googlemaps()

Template for googlemaps.html implementing Overlay of tiles for 'mercator' profile. It returns filled string. Expected variables: title, googlemapskey, north, south, east, west, minzoom, maxzoom, tilesize, tileformat, publishurl

generate_leaflet()

Template for leaflet.html implementing overlay of tiles for 'mercator' profile. It returns filled string. Expected variables: title, north, south, east, west, minzoom, maxzoom, tilesize, tileformat, publishurl

generate_metadata()

Generation of main metadata files and HTML viewers (metadata related to particular tiles are generated during the tile processing).

generate_openlayers()

Template for openlayers.html implementing overlay of available Spherical Mercator layers.

It returns filled string. Expected variables: title, bingkey, north, south, east, west, minzoom, maxzoom, tilesize, tileformat, publishurl

generate_tilemapresource()

Template for tilemapresource.xml. Returns filled string. Expected variables: title, north, south, east, west, isepsg4326, projection, publishurl, zoompixels, tilesize, tileformat, profile

geo_query(*ds*, *ulx*, *uly*, *lrx*, *lry*, *querysize=0*)
For given dataset and query in cartographic coordinates returns parameters for ReadRaster() in raster coordinates and x/y shifts (for border tiles). If the querysize is not given, the extent is returned in the native resolution of dataset ds.

raises Gdal2TilesError if the dataset does not contain anything inside this geo_query

open_input()
Initialization of the input raster, reprojection if necessary

exception gdal2tiles.gdal2tiles.**GDALError**
Bases: Exception

exception gdal2tiles.gdal2tiles.**Gdal2TilesError**
Bases: Exception

class gdal2tiles.gdal2tiles.**GlobalGeodetic**(*tmscompatible*, *tileSize=256*)
Bases: object

Functions necessary for generation of global tiles in Plate Carre projection, EPSG:4326, "unprojected profile".

Such tiles are compatible with Google Earth (as any other EPSG:4326 rasters) and you can overlay the tiles on top of OpenLayers base map.

Pixel and tile coordinates are in TMS notation (origin [0,0] in bottom-left).

What coordinate conversions do we need for TMS Global Geodetic tiles?

Global Geodetic tiles are using geodetic coordinates (latitude,longitude) directly as planar coordinates XY (it is also called Unprojected or Plate Carre). We need only scaling to pixel pyramid and cutting to tiles. Pyramid has on top level two tiles, so it is not square but rectangle. Area [-180,-90,180,90] is scaled to 512x256 pixels. TMS has coordinate origin (for pixels and tiles) in bottom-left corner. Rasters are in EPSG:4326 and therefore are compatible with Google Earth.

LatLon <-> Pixels <-> Tiles

WGS84 coordinates Pixels in pyramid Tiles in pyramid

lat/lon XY pixels Z zoom XYZ from TMS

EPSG:4326 .—-. —-

/ <-> /——/ <-> TMS / /————/

—— /———————/

WMS, KML Web Clients, Google Earth TileMapService

LonLatToPixels(*lon*, *lat*, *zoom*)
Converts lon/lat to pixel coordinates in given zoom of the EPSG:4326 pyramid

LonLatToTile(*lon*, *lat*, *zoom*)
Returns the tile for zoom which covers given lon/lat coordinates

PixelsToTile(*px*, *py*)
Returns coordinates of the tile covering region in pixel coordinates

Resolution(*arc/pixel) for given zoom level (measured at Equator*)

TileBounds(*tx*, *ty*, *zoom*)
Returns bounds of the given tile

TileLatLonBounds(*tx*, *ty*, *zoom*)
Returns bounds of the given tile in the SWNE form

ZoomForPixelSize(*pixelSize*)
Maximal scaledown zoom of the pyramid closest to the pixelSize.

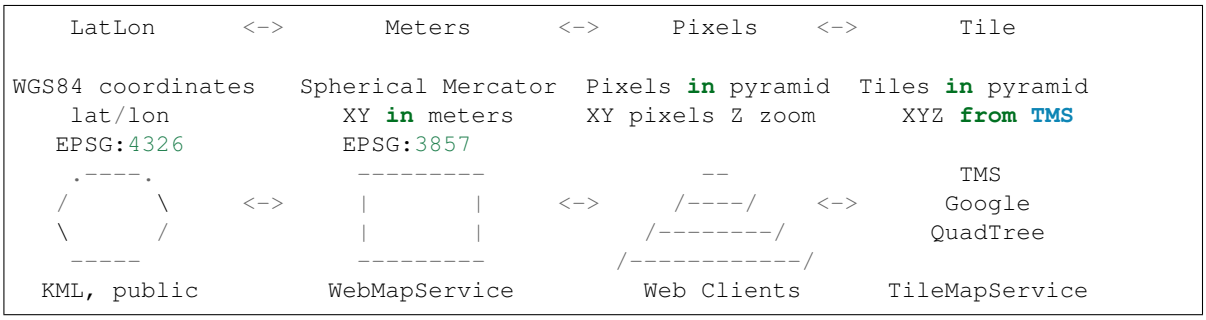
class gdal2tiles.gdal2tiles.**GlobalMercator**(*tileSize=256*)
Bases: object

Functions necessary for generation of tiles in Spherical Mercator projection, EPSG:3857.

Such tiles are compatible with Google Maps, Bing Maps, Yahoo Maps, UK Ordnance Survey OpenSpace API, ... and you can overlay them on top of base maps of those web mapping applications.

Pixel and tile coordinates are in TMS notation (origin [0,0] in bottom-left).

What coordinate conversions do we need for TMS Global Mercator tiles:



What is the coordinate extent of Earth in EPSG:3857?

[-20037508.342789244, -20037508.342789244, 20037508.342789244, 20037508.342789244] Constant 20037508.342789244 comes from the circumference of the Earth in meters, which is 40 thousand kilometers, the coordinate origin is in the middle of extent. In fact you can calculate the constant as: 2 * math.pi * 6378137 / 2.0 $ echo 180 85 | gdaltransform -s_srs EPSG:4326 -t_srs EPSG:3857 Polar areas with abs(latitude) bigger then 85.05112878 are clipped off.

What are zoom level constants (pixels/meter) for pyramid with EPSG:3857?

whole region is on top of pyramid (zoom=0) covered by 256x256 pixels tile, every lower zoom level resolution is always divided by two initialResolution = 20037508.342789244 * 2 / 256 = 156543.03392804062

What is the difference between TMS and Google Maps/QuadTree tile name convention?

The tile raster itself is the same (equal extent, projection, pixel size), there is just different identification of the same raster tile. Tiles in TMS are counted from [0,0] in the bottom-left corner, id is XYZ. Google placed the origin [0,0] to the top-left corner, reference is XYZ. Microsoft is referencing tiles by a QuadTree name, defined on the website: http://msdn2.microsoft.com/en-us/library/bb259689.aspx

The lat/lon coordinates are using WGS84 datum, yes?

Yes, all lat/lon we are mentioning should use WGS84 Geodetic Datum. Well, the web clients like Google Maps are projecting those coordinates by Spherical Mercator, so in fact lat/lon coordinates on sphere are treated as if the were on the WGS84 ellipsoid.

From MSDN documentation: To simplify the calculations, we use the spherical form of projection, not the ellipsoidal form. Since the projection is used only for map display, and not for displaying numeric coordinates, we don't need the extra precision of an ellipsoidal projection. The spherical projection causes approximately 0.33 percent scale distortion in the Y direction, which is not visually noticeable.

How do I create a raster in EPSG:3857 and convert coordinates with PROJ.4?

You can use standard GIS tools like gdalwarp, cs2cs or gdaltransform. All of the tools supports -t_srs 'epsg:3857'.

For other GIS programs check the exact definition of the projection: More info at http://spatialreference.org/ref/user/google-projection/ The same projection is designated as EPSG:3857. WKT definition is in the official EPSG database.

Proj4 Text: +proj=merc +a=6378137 +b=6378137 +lat_ts=0.0 +lon_0=0.0 +x_0=0.0 +y_0=0 +k=1.0 +units=m +nadgrids=@null +no_defs

Human readable WKT format of EPSG:3857:

PROJCS["Google Maps Global Mercator",

GEOGCS["WGS 84",

DATUM["WGS_1984",

SPHEROID["WGS 84",6378137,298.257223563, AUTHORITY["EPSG","7030"]],

AUTHORITY["EPSG","6326"]],

PRIMEM["Greenwich",0], UNIT["degree",0.0174532925199433], AUTHORITY["EPSG","4326"]],

PROJECTION["Mercator_1SP"], PARAMETER["central_meridian",0], PARAMETER["scale_factor",1], PARAMETER["false_easting",0], PARAMETER["false_northing",0], UNIT["metre",1,

AUTHORITY["EPSG","9001"]]]

GoogleTile(*tx*, *ty*, *zoom*)
Converts TMS tile coordinates to Google Tile coordinates

LatLonToMeters(*lat*, *lon*)
Converts given lat/lon in WGS84 Datum to XY in Spherical Mercator EPSG:3857

MetersToLatLon(*mx*, *my*)
Converts XY point from Spherical Mercator EPSG:3857 to lat/lon in WGS84 Datum

MetersToPixels(*mx*, *my*, *zoom*)
Converts EPSG:3857 to pyramid pixel coordinates in given zoom level

MetersToTile(*mx*, *my*, *zoom*)
Returns tile for given mercator coordinates

PixelsToMeters(*px*, *py*, *zoom*)
Converts pixel coordinates in given zoom level of pyramid to EPSG:3857

PixelsToRaster(*px*, *py*, *zoom*)
Move the origin of pixel coordinates to top-left corner

PixelsToTile(*px*, *py*)
Returns a tile covering region in given pixel coordinates

QuadTree(*tx*, *ty*, *zoom*)
Converts TMS tile coordinates to Microsoft QuadTree

Resolution(*meters/pixel) for given zoom level (measured at Equator*)

TileBounds(*tx*, *ty*, *zoom*)
Returns bounds of the given tile in EPSG:3857 coordinates

TileLatLonBounds(*tx*, *ty*, *zoom*)
Returns bounds of the given tile in latitude/longitude using WGS84 datum

ZoomForPixelSize(*pixelSize*)
Maximal scaledown zoom of the pyramid closest to the pixelSize.

class gdal2tiles.gdal2tiles.**ProgressBar**(*total_items*)
Bases: object

log_progress(*nb_items=1*)

start()

class gdal2tiles.gdal2tiles.**TileDetail**(***kwargs*)
Bases: object

querysize = 0

rx = 0

rxsize = 0

ry = 0

rysize = 0

tx = 0

ty = 0

tz = 0

wx = 0

wxsize = 0

wy = 0

wysize = 0

class gdal2tiles.gdal2tiles.**TileJobInfo**(***kwargs*)
Bases: object

Plain object to hold tile job configuration for a dataset

in_srs_wkt = 0

is_epsg_4326 = False

kml = False

nb_data_bands = 0

ominy = 0

options = None

out_geo_trans = []

output_file_path = ''

src_file = ''

tile_driver = None

tile_extension = ''

tile_size = 0

tmaxz = 0

tminmax = []

tminz = 0

class gdal2tiles.gdal2tiles.**Zoomify**(*width*, *height*, *tilesize=256*, *tileformat='jpg'*)
Bases: object

tilefilename(*x*, *y*, *z*)
Returns filename for tile with given coordinates

gdal2tiles.gdal2tiles.**add_alpha_band_to_string_vrt**(*vrt_string*)

gdal2tiles.gdal2tiles.**add_gdal_warp_options_to_string**(*vrt_string*, *warp_options*)

gdal2tiles.gdal2tiles.**create_base_tile**(*tile_job_info*, *tile_detail*, *queue=None*)

gdal2tiles.gdal2tiles.**create_overview_tiles**(*tile_job_info*, *output_folder*, *options*)
Generation of the overview tiles (higher in the pyramid) based on existing tiles

gdal2tiles.gdal2tiles.**exit_with_error**(*message*, *details=''*)

gdal2tiles.gdal2tiles.**generate_kml**(*tx*, *ty*, *tz*, *tileext*, *tilesize*, *tileswne*, *options*, *children=None*, ***args*)
Template for the KML. Returns filled string.

gdal2tiles.gdal2tiles.**generate_tiles**(*input_file*, *output_folder*, ***options*)
Generate tiles from input file.

Arguments: input_file (str): Path to input file.

output_folder (str): Path to output folder.

options: Tile generation options.

Options:

profile (str): Tile cutting profile (mercator,geodetic,raster) - default 'mercator' (Google Maps compatible)

resampling (str): Resampling method (average,near,bilinear,cubic,cubicsp line,lanczos,antialias) - default 'average'

s_srs: The spatial reference system used for the source input data

zoom: Zoom levels to render; format: *[int min, int max],* *'min-max'* or *int/str zoomlevel*.

tile_size (int): Size of tiles to render - default 256

resume (bool): Resume mode. Generate only missing files.

srcnodata: NODATA transparency value to assign to the input data

tmscompatible (bool): When using the geodetic profile, specifies the base resolution as 0.703125 or 2 tiles at zoom level 0.

verbose (bool): Print status messages to stdout

kml (bool): Generate KML for Google Earth - default for 'geodetic' profile and 'raster' in EPSG:4326. For a dataset with different projection use with caution!

url (str): URL address where the generated tiles are going to be published

webviewer (str): Web viewer to generate (all,google,openlayers,none) - default 'all'

title (str): Title of the map

copyright (str): Copyright for the map

googlekey (str): Google Maps API key from http://code.google.com/apis/maps/signup.html

bingkey (str): Bing Maps API key from https://www.bingmapsportal.com/

nb_processes: Number of processes to use for tiling.

gdal2tiles.gdal2tiles.**get_tile_swne**(*tile_job_info*, *options*)

gdal2tiles.gdal2tiles.**gettempfilename**(*suffix*)
Returns a temporary filename

gdal2tiles.gdal2tiles.**has_georeference**(*dataset*)

gdal2tiles.gdal2tiles.**multi_threaded_tiling**(*input_file*, *output_folder*, ***options*)
Generate tiles with multi processing.

gdal2tiles.gdal2tiles.**nb_data_bands**(*dataset*)
Return the number of data (non-alpha) bands of a gdal dataset

gdal2tiles.gdal2tiles.**options_post_processing**(*options*, *input_file*, *output_folder*)

gdal2tiles.gdal2tiles.**process_options**(*input_file*, *output_folder*, *options={}*)

gdal2tiles.gdal2tiles.**progress_printer_thread**(*queue*, *nb_jobs*)

gdal2tiles.gdal2tiles.**reproject_dataset**(*from_dataset*, *from_srs*, *to_srs*, *options=None*)
Returns the input dataset in the expected "destination" SRS. If the dataset is already in the correct SRS, returns it unmodified

gdal2tiles.gdal2tiles.**scale_query_to_tile**(*dsquery*, *dstile*, *tiledriver*, *options*, *tilefilename=''*)
Scales down query dataset to the tile dataset

gdal2tiles.gdal2tiles.**setup_input_srs**(*input_dataset*, *options*)
Determines and returns the Input Spatial Reference System (SRS) as an osr object and as a WKT representation

Uses in priority the one passed in the command line arguments. If None, tries to extract them from the input dataset

gdal2tiles.gdal2tiles.**setup_no_data_values**(*input_dataset*, *options*)
Extract the NODATA values from the dataset or use the passed arguments as override if any

gdal2tiles.gdal2tiles.**setup_output_srs**(*input_srs*, *options*)
Setup the desired SRS (based on options)

gdal2tiles.gdal2tiles.**single_threaded_tiling**(*input_file*, *output_folder*, ***options*)
Generate tiles using single process.

Keep a single threaded version that stays clear of multiprocessing, for platforms that would not support it

gdal2tiles.gdal2tiles.**update_alpha_value_for_non_alpha_inputs**(*warped_vrt_dataset*, *options=None*)
Handles dataset with 1 or 3 bands, i.e. without alpha channel, in the case the nodata value has not been forced by options

gdal2tiles.gdal2tiles.**update_no_data_values**(*warped_vrt_dataset*, *nodata_values*, *options=None*)
Takes an array of NODATA values and forces them on the WarpedVRT file dataset passed

gdal2tiles.gdal2tiles.**worker_tile_details**(*input_file*, *output_folder*, *options*, *send_pipe=None*)

3.1.3 gdal2tiles.utils module

class gdal2tiles.utils.**AttrDict**(*d={}*)
Bases: object

Helper class to provide attribute like access (read and write) to dictionaries. Used to provide a convenient way to access both results and nested dsl dicts.

to_dict()

gdal2tiles.utils.**recursive_attrdict**(*obj*)
Deprecated since version version.

Walks a simple data structure, converting dictionary to AttrDict. Supports lists, tuples, and dictionaries.

3.1.4 Module contents

Top-level package for gdal2tiles library.

CHAPTER 4

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

4.1 Types of Contributions

4.1.1 Report Bugs

Report bugs at https://github.com/tehamalab/gdal2tiles/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

4.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

4.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

4.1.4 Write Documentation

gdal2tiles could always use more documentation, whether as part of the official gdal2tiles docs, in docstrings, or even on the web in blog posts, articles, and such.

4.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/tehamalab/gdal2tiles/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

4.2 Get Started!

Ready to contribute? Here's how to set up *gdal2tiles* for local development.

1. Fork the *gdal2tiles* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/gdal2tiles.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv gdal2tiles
$ cd gdal2tiles/
$ pip install -e .
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 gdal2tiles tests
$ python setup.py test or py.test
$ tox
```

   To get flake8, tox and other development dependancies you can install them on your virtual environment using pip.

   $ pip istall -r requirements_dev.txt

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

4.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. It is highly recommended for the pull request to include relevant tests whenever applicable.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 3.6, 3.7 and 3.8. Check https://travis-ci.org/tehamalab/gdal2tiles/pull_requests and make sure that the tests pass for all supported Python versions.

4.4 Tips

To run a subset of tests:

```
$ py.test tests.test_gdal2tiles
```

4.5 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ bumpversion patch # possible: major / minor / patch
$ git push
$ git push --tags
```

CHAPTER 5

Credits

5.1 Development Lead

- Tehamalab <developers@tehamalab.com>

5.2 Contributors

None yet. Why not be the first?

CHAPTER 6

History

6.1 0.1.9 (2020-09-30)

- Enabling GDAL Exceptions.
- Ensuring `tmscompatible` option in tile generation and calculation is bool instance.

6.2 0.1.8 (2020-09-23)

- Fix AttributeError `gdal2tiles.generate_tiles(..., profile='raster', kml=True)`. Fix issue #14.

6.3 0.1.7 (2020-06-03)

- Add tile_size option on *generate_tiles* to allow custom tile sizes.
- Small documentation updates.
- Improve basic tests and test against multiple versions of GDAL.

6.4 0.1.6 (2020-01-03)

- Fix some of GDAL installation issues.
- Use pygdal package instead of GDAL dependency.

6.5 0.1.5 (2018-08-14)

- Bug fix.

6.6 0.1.4 (2018-08-14)

- Accept list or tuple in specifying tile generation zoom level.

6.7 0.1.3 (2018-07-31)

- Use billard for multiprocessing if available.

6.8 0.1.2 (2018-05-16)

- Bug fix in `generate_tiles()`.

6.9 0.1.1 (2018-05-10)

- Clean the source code.
- Setup documentation
- Setup testing environment

6.10 0.1.0 (2018-05-06)

- First release on PyPI.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

g

Index

W

Z