HYLAS

Release 0.0.1

Sebastian Schwindt

Oct 29, 2020

CONTENTS

hylas extracts geo-spatial information from *las* files and converts them to ESRI shapefiles or GeoTIFF rasters. *las* is the typical file format for storing airborne lidar (Light Detection and Ranging) data. *hylas* is a *Python3* package and this documentation uses a *Sphinx readthedocs* theme. The functional core of *hylas* involves the creation of:

- A point shapefile with user-defined point attributes such as *intensity*, *waveform*, or *nir*.
- Digital elevation model (DEM) with user-defined resolution (pixel size).
- *GeoTIFF* rasters with user-defined resolution (pixel size) for any attribute of a *las* file (e.g., *intensity*, *waveform*, or *nir*).

To ensure the best experience with *hylas* and its useful functions, follow the *Installation* instructions for setting up the working environment either on *Linux* or on *Windows*.

Get the *hylas* docs as PDF

This documentation is also as available as a style-adapted PDF (download).

CHAPTER

ONE

INSTALLATION

1.1 LINUX (Debian/Ubuntu)

1.1.1 Optional: Use a Virtual Machine (VM)

Either download a net-installer *ISO* of Debian Linux or Ubuntu, or use the OSGeoLive, and install one of theses images as a Virtual Machine (VM). To get started with VMs read the introduction to VMs on hydro-informatics.github.io. Installing the *OSGeoLive* VM works similar, as described on hydro-informatics.github.io, but use the *OSGeoLive* image in lieu of the *Debian Linux ISO*. After installing *Linux* as a VM, make sure to:

- Install Guest Additions for *Linux* VMs in *VirtualBox*.
- Enable folder sharing between the host and guest (*Debian*, *Ubuntu*, or *OSGeoLive* image).

Other system setups described on hydro-informatics.github.io (e.g., *Wine*) are not required in the following.

1.1.2 Prepare your system

Open *Terminal* and update the system:

```
sudo apt update && sudo apt full-upgrade -y
```

1.1.3 Update Python references

Most *Linux* distributions still have *Python2* implemented as base interpreter to be used when `python` is called in *Terminal*. However, *Python2* usage is deprecated, and therefore, we want to make sure to robustly use *Python3* for running any *Python* script. Check out the installed *Python3* versions:

```
ls /usr/bin/python*


 /usr/bin/python  /usr/bin/python2  /usr/bin/python2.7  /usr/bin/python3  /usr/bin/
↪python3.8  /usr/bin/python3.8m  /usr/bin/python3m
```

In this example, *Python2.7* and *Python3.8* are installed. To overwrite *Python2* usage, set the `python` environment variable so that it points at *Python3*:

```
sudo update-alternatives --install /usr/bin/python python /usr/bin/python3.6 2
alias python=python3
```

1.1.4 PIP3 and additional libraries for geospatial analysis

Make sure that PyGeos and tkinter are available for use with geopandas:

```
sudo apt install python3-pip
sudo apt-get install python3-tk
sudo apt install tk8.6-dev
sudo apt install libgeos-dev
```

Then install *QGIS* and `GDAL` for *Linux* (this should work for any *Debian* architecture):

```
sudo add-apt-repository ppa:ubuntugis/ppa && sudo apt-get update
sudo apt-get update
sudo apt-get install gdal-bin
sudo apt-get install libgdal-dev
export CPLUS_INCLUDE_PATH=/usr/include/gdal
export C_INCLUDE_PATH=/usr/include/gdal
pip3 install GDAL
```

Note: Check on the latest GDAL release on the developers website.

More guidance for installing GDAL (also on other platforms) is available at gdal.org.

1.1.5 Install an IDE (*PyCharm*)

Note: IDE - your choice Any other Python IDE is also OK for working with *hylas*. Setting up PyCharm is explained here as just one option for working with *hylas*.

Install *PyCharm* with snap (requires snap):

```
sudo apt install snapd
sudo snap install pycharm-community --classic
```

1.2 WINDOWS

1.2.1 Required software

On *Windows*, a convenient option for working with *hylas* is to use a conda environment. In addition, *GitBash* is necessary to clone (download) *hylas* (and to keep posted on updates). In detail:

- Install *Anaconda*, for example, as described on hydro-informatics.github.io.
- Download and install *GitBash*.

1.2.2 Create a conda environment

Then open *Anaconda Prompt* and create a new environment (e.g., `ipy-hylas`):

```
conda env create --name ipy-hylas python=3.8
```

Then, activate the new environment:

```
conda activate ipy-hylas
```

Install the required Python libraries in the new environment:

```
conda update conda
conda install -c anaconda numpy
conda install -c anaconda pandas
conda install -c conda-forge gdal
conda install -c conda-forge shapely
conda install -c conda-forge alphashape
conda install -c conda-forge rasterstats
conda install -c anaconda scikit-image
conda install -c conda-forge geopandas
conda install -c conda-forge laspy
```

There are more compact ways to setup the conda environment (e.g., using an environment file). To read more about conda environments go to hydro-informatics.github.io.

1.2.3 Install an IDE (*PyCharm*)

Note: IDE - your choice Any other Python IDE is also OK for working with *hylas*. Setting up PyCharm is explained here as just one option for working with *hylas*.

Download and install *PyCharm Community Edition*. Read more at hydro-informatics.github.io.

1.3 GET HYLAS

1.3.1 Clone hylas

Open *Terminal* (or *Anaconda Prompt*), create a project folder, and `cd` to the project folder:

```
mkdir hylas-project
cd hylas-project
```

Clone the *hylas* repository in the new folder:

```
git clone https://github.com/sschwindt/lidar-analysis.git
```

Note: Cloning the repository creates a new sub-folder. So if you want to work directly in your home folder, skip the `mkdir` + `cd` commands.

LINUX / PIP3 USERS

In *Terminal* `cd` to the local *hylas* repository to install and update (upgrade) required Python packages:

```
pip3 install -r requirements.txt
pip3 install -r requirements.txt --upgrade
```

Clean up obsolete update remainders:

```
sudo apt-get clean
sudo apt-get autoclean
sudo apt-get autoremove
sudo apt-get autoremove --purge
```

Windows / conda users can skip the installation of requirements, because those were already installed in the *conda* environment.

1.3.2 Setup *PyCharm* IDE

Start *PyCharm* and create a new project from the `hylas` repository:

- Open *PyCharm*, click on `+ Create New Project` and select the directory where you cloned *hylas* (e.g., `/ROOT/git/hylas`).
- Define a *Project Interpreter* depending on if you use *Linux / pip3* or *Windows / *Anaconda*. So choose `New` > `Add Python Interpreter`

LINUX / PIP3 USERS

Make sure to use the system interpreter `/usr/bin/python3` (*Project* > *Settings* > *Interpreter*). You will probably get a warning message about using the system interpreter for a project, but this is acceptable when you are working on a VM.

WINDOWS / ANACONDA USERS

- Enable the *View hidden folders* option to see the `AppData` folder in *Windows Explorer*. *Microsoft* explains how this works on their support website. Then, you can copy-paste folder directories from *Windows Explorer* to *PyCharm*.
- Identify the system path where the conda environment (e.g. `ipy-hylas`) lives. Typically, this is something like `C:\users\<your-user-name>\AppData\Local\Continuum\anaconda3\envs\ipy-hylas`.
- In the **Add Python Interpreter** window, go to the *Conda Environment* tab, select *New environment*, and make the following settings:
  - Location: `C:\users\<your-user-name>\AppData\Local\Continuum\anaconda3\envs\ipy-hylas`
  - Python version: `3.8`
  - Conda executable: `C:\users\<your-user-name>\AppData\Local\Continuum\anaconda3\bin\conda`

There is also a detailed tutorial for setting up *PyCharm* with *Anaconda* available at docs.anaconda.com.

CHAPTER

TWO

USAGE

2.1 Basic usage

To convert a *las* file to an ESRI shapefile or GeoTIFF, load *hylas* in Python from the directory where you downloaded (cloned) *hylas*:

```
import hylas
las_file_name = "path/to/a/las-file.las"
methods = ["las2shp", "las2tif"]
hylas.process_file(las_file_name, epsg=3857, methods=methods)
```

The above code block defines a `las_file_name` variable and `methods` to be used with `hylas.process_file` (see *The main file: hylas.py*). The function accepts many more optional arguments:

Loads a las-file and convert it to another geospatial file format (keyword arguments `**opts`).

param source_file_name Full directory of the source file to use with methods * if method="las2*" > provide a las-file name * if method="shp2*" > provide a shapefile name

type source_file_name str

param epsg Authority code to use (try `hylas.lookup_epsg(las_file_name)` to look up the epsg online).

type epsg int

keyword create_dem default: False - set to True for creating a digital elevation model (DEM)

kwtype create_dem bool

keyword extract_attributes Attributes to extract from the las-file available in pattr (config.py)

kwtype extract_attributes str

keyword methods Enabled list strings are las2shp, las2tif, shp2tif, las2dem

kwtype methods *list* [*str*]

keyword overwrite Overwrite existing shapefiles and/or GeoTIFFs (default: `True`).

kwtype overwrite bool

keyword pixel_size Use with *2tif to set the size of pixels relative to base units (pixel_size=5 > 5-m pixels)

kwtype pixel_size float

keyword shapefile_name Name of the point shapefile to produce with las2*

kwtype shapefile_name str

keyword tif_prefix Prefix include folder path to use for GeoTiFFs (defined extract_attributes are appended to file name)

kwtype tif_prefix str

keyword interpolate_gap_pixels Fill empty pixels that are not touched by a shapefile point with interpolated values (default: `True`)

kwtype interpolate_gap_pixels bool

keyword radius1 Define the x-radius for interpolating pixels (default: `-1`, corresponding to infinity). Only applicable `with interpolate_gap_pixels`.

kwtype radius1 float

keyword radius2 Define the y-radius for interpolating pixels (default: `-1`, corresponding to infinity). Only applicable `with interpolate_gap_pixels`.

kwtype radius2 float

keyword power Power of the function for interpolating pixel values (default: `1.0`, corresponding to linear).

kwtype power float

keyword smoothing Smoothing parameter for interpolating pixel values (default: `0.0`).

kwtype smoothing float

keyword min_points Minimum number of points to use for interpolation. If the interpolator cannot find at least `min_points` for a pixel, it assigns a `no_data` value to that pixel (default: `0`).

kwtype min_points int

keyword max_points Maximum number of points to use for interpolation. The interpolator will not use more than `max_points` closest points to interpolate a pixel value (default: `0`).

kwtype max_points int

returns `True` if successful, `False` otherwise

rtype bool

Note: The `LasPoint` class (see *The LasPoint class*) can also be directly called in any script with `hylas.LasPoint`. Have a look at the `hylas.process_file` function (*The main file: hylas.py*) to see how an instance of the `LasPoint` class is used.

2.2 Application example

The file `ROOT/test.py` provides and example for using `hylas` with a las-file stored in a new folder `ROOT/data`:

```
import hylas
import os

las_file_name = os.path.abspath("") + "/data/las-example.las"
shp_file_name = os.path.abspath("") + "/data/example.shp"
epsg = 25832
methods = ["las2tif"]
attribs = "aci"
```

(continues on next page)

(continued from previous page)

```
px_size = 2
tif_prefix = os.path.abspath("") + "/data/sub"

hylas.process_file(las_file_name,
                   epsg=epsg,
                   methods=methods,
                   extract_attributes=attribs,
                   pixel_size=px_size,
                   shapefile_name=shp_file_name,
                   tif_prefix=tif_prefix)
```

Note: The method `las2tif` automatically calls the `las2shp` (`hylas.LasPoint.export2shp`) method because the GeoTIFF pixel values are extracted from the attribute table of the point shapefile. So `las2shp` is the baseline for any other operation.

2.3 Geo-utils

The implemented `geo_utils` package is forked from hydro-informatics on *GitHub*. `geo_utils` provides routines for creating, modifying, and transforming geo-spatial datasets. A detailed documentation of `geo_utils` is available at geo-utils.readthedocs.io.”””

to enable creating correctly geo-referenced GeoTIFF rasters (`rasterize` function - see *geo_utils*).

CHAPTER

THREE

CODE DOCUMENTATION

3.1 The main file: hylas.py

`hylas.lookup_epsg(file_name)`

Starts a google search to retrieve information from a file name (or other `str`) with information such as *UTM32*.

Parameters **file_name** (`str`) – file name or other string with words separated by "-" or "_"

Notes

- This function opens a google search in the default web browser.
- More information about projections, spatial reference systems, and coordinate systems

  can be obtained with the geo_utils package.

```
process_file(source_file_name, epsg, **opts)
```

Loads a las-file and convert it to another geospatial file format (keyword arguments `**opts`).

Note that this function documentation is currently manually implemented because of *Sphinx* having troubles to look behind decorators.

Arguments:

- **source_file_name (`str`): Full directory of the source file to use with methods**
  - if `method="las2*"`: provide a las-file name
  - if `method="shp2*"`: provide a shapefile name
- **epsg** (`int`): Authority code to use (try `hylas.lookup_epsg(las_file_name)` to look up the epsg online).

Keyword Arguments (`**opts`):

- **create_dem** (`bool`): Set to True for creating a digital elevation model (DEM - default: `False`)
- **extract_attributes** (`str`): Attributes to extract from the las-file available in `pattr` (`config.py`)
- **methods** (`list [str]`): Enabled list strings are `las2shp`, `las2tif`, `shp2tif`, `las2dem`
- **overwrite** (`bool`): Overwrite existing shapefiles and/or GeoTIFFs (default: `True`).
- **pixel_size** (`float`): Use with *2tif to set the size of pixels relative to base units (`pixel_size=5` indicates 5x5-m pixels)

- **shapefile_name** (`str`): Name of the point shapefile to produce with `las2*`
- **tif_prefix** (`str`): Prefix include folder path to use for GeoTiFFs (defined extract_attributes are appended to file name)
- **interpolate_gap_pixels** (`bool`): Fill empty pixels that are not touched by a shapefile point with interpolated values (default: `True`)
- **radius1** (`float`): Define the x-radius for interpolating pixels (default: `-1`, corresponding to infinity). Only applicable `with interpolate_gap_pixels`.
- **radius2** (`float`): Define the y-radius for interpolating pixels (default: `-1`, corresponding to infinity). Only applicable `with interpolate_gap_pixels`.
- **power** (`float`): Power of the function for interpolating pixel values (default: `1.0`, corresponding to linear).
- **smoothing** (`float`): Smoothing parameter for interpolating pixel values (default: `0.0`).
- **min_points** (`int`): Minimum number of points to use for interpolation. If the interpolator cannot find at least `min_points` for a pixel, it assigns a `no_data` value to that pixel (default: `0`).
- **max_points** (`int`): Maximum number of points to use for interpolation. The interpolator will not use more than `max_points` closest points to interpolate a pixel value (default: `0`).

Returns: `bool`: `True` if successful, `False` otherwise.

More information on pixel value interpolation: * `interpolate_gap_pixels=True` interpolates values at pixels that are not touched by any las point. * The pixel value interpolation uses `gdal_grid` (i.e., its Python bindings through `gdal.Grid()`). * Control the interpolation parameters with the keyword arguments `radius1`, `radius2`, `power`, `max_points`, `min_points`, and `smoothing`.

See also:

All variables are illustratively explained on the GDAL website.

3.2 Basic parameters: config.py

This is the hylas config file - do not confuse with docs/conf (sphinx documentation config) nor geo_utils/geoconfig.

`config.`**`pattr = {'C': 'classification', 'G': 'gps_time', 'N': 'num_returns', 'R': 'return_nu`**

`dict`: `dict` of attributes to extract data layers (shapefile columns or multiple GeoTIFFs) from a las file.

All attributes defined in `pattr.values()` must be an attribute of a las_file object. Print all available las file attributes with:

```
print(dir(LasPoint.las_file))
```

`config.`**`wattr = {'C': 'Class', 'G': 'GPStime', 'N': 'NumberRet', 'R': 'ReturnNumber', 'W':`**

`dict`: `dict` with column headers (shapefile attribute table) and GeoTIFF file names to use for parsing attributes.

3.3 Global functions: helpers.py

helpers.**cache**(*fun*)
Makes a function running in a __cache__ sub-folder to enable deleting temporary trash files.

helpers.**check_cache**()
Creates the cache folder if it does not exist.

helpers.**dict2str**(*dictionary*, ***kwargs*)
Converts a dict to a string expression.

Parameters **dictionary** (*dict*) – A dictionary to convert to a string.

Keyword Arguments **inverse_dict** (*boolean*) – Apply inverse order of string (default: `False`).

Returns The dictionary as flattened text.

Return type str

Example

```
>>> dict2str({e:  1, f:  2, ...}) Out[]:  "{e:  1, f:  2, ...}"
```

helpers.**log_actions**(*fun*)
Wraps a function with a logger. All actions of the wrapped function can be writting to `ROOT/logfile.log`.

helpers.**remove_directory**(*directory*)
Removes a directory and all its contents - be careful!

Parameters **directory** (*str*) – directory to remove (delete)

Returns Deletes directory.

Return type None

helpers.**start_logging**()
Creates a log file (`ROOT/logfile.log`).

3.4 The `LasPoint` class

class LasPoint.**LasPoint**(*las_file_name*, *epsg=3857*, *use_attributes='aciw'*, *overwrite=True*)
Las file container to convert datasets to ESRI point shapefiles and/or GeoTIFFs.

Parameters

- **las_file_name** (*str*) – Directory to and name of a las file.
- **epsg** (*int*) – Authority Code - Geodetic Parameter Dataset ID (default: `3857`).
- **overwrite** (*bool*) – Overwrite existing shapefiles and/or GeoTIFFs (default: `True`).
- **use_attributes** (*str*) – Attributes (properties) to use from the las-file available in pattr (config.py). (default: `use_attributes="aciw"`).

Variables

- **las_file** (*laspy.file.File*) – A laspy file object
- **attributes** (*str*) – Defined with `use_attributes`

- **epsg** (*int*) – Authority code
- **gdf** (*geopandas.GeoDataFrame*) – geopandas data frame containing all points of the las file with the properties (columns) defined by `use_attributes`
- **offset** (*laspy.file.File()header.offset*) – Offset of las points (auto-read)
- **overwrite** (*bool*) – Enable or disable overwriting existing files (default: `True`)
- **scale** (*laspy.file.File()header.scale*) – Scale of las points relative to the offset (auto-read)
- **shapefile_name** (*str*) – The name and dicrectorty of a point shapefile where all las-file data is stored
- **srs** (*osr.SpatialReference*) – The geo-spatial reference imported from `epsg`

_build_data_frame()
Builds the geopandas GeoDataFrame - auto-runs `self._parse_attributes`.

_get_xyz_array()
Extract x-y-z data from las records in a faster way than using `las_file.x`, `y`, or `z`.

Returns The DEM information extracted from the las file.

Return type ndarray

_parse_attributes()
Parses attributes and append entries to point list.

create_dem(*target_file_name=''*, *pixel_size=1.0*, ***kwargs*)
Creates a digital elevation model (DEM) in GeoTIFF format from the *las* file points.

Parameters

- **target_file_name** (*str*) – A file name including an existing directory where the dem will be created< must end on `.tif`.
- **pixel_size** (*float*) – The size of one pixel relative to the spatial reference system

Keyword Arguments

- **src_shp_file_name** (*str*) – Name of a shapefile from which elevation information is to be extracted (default: name of the las-point shapefile)
- **elevation_field_name** (*str*) – Name of the field from which elevation data is to be extracted (default: `"elevation"`)
- **interpolate_gap_pixels** (*bool*) – Fill empty pixels that are not touched by a shapefile point with interpolated values (default: `False`)
- **radius1** (*float*) – Define the x-radius for interpolating pixels (default: `-1`, corresponding to infinity). Only applicable `with interpolate_gap_pixels`.
- **radius2** (*float*) – Define the y-radius for interpolating pixels (default: `-1`, corresponding to infinity). Only applicable `with interpolate_gap_pixels`.
- **power** (*float*) – Power of the function for interpolating pixel values (default: `1.0`, corresponding to linear).
- **smoothing** (*float*) – Smoothing parameter for interpolating pixel values (default: `0.0`).
- **min_points** (*int*) – Minimum number of points to use for interpolation. If the interpolator cannot find at least `min_points` for a pixel, it assigns a `no_data` value to that pixel (default: `0`).

- **max_points** (*int*) – Maximum number of points to use for interpolation. The interpolator will not use more than `max_points` closest points to interpolate a pixel value (default: `0`).

Hint: This function works independently and does not require the prior creation of a shapefile.

Returns `0` if successful, otherwise `-1`

Return type int

export2shp(***kwargs*)

Converts las file points to a point shapefile.

Keyword Arguments **shapefile_name** (*str*) – Optional shapefile name (must end on .shp). (default: `'/this/dir/las_file_name.shp'`).

Returns `/path/to/shapefile.shp`, which is a point shapefile created by the function.

Return type str

get_file_info()

Prints las file information to console.

3.5 geo_utils

3.5.1 geo_utils MASTER (geo_utils.py)

`geo_utils` is a package for creating, modifying, and transforming geo-spatial datasets. A detailed documentation of `geo_utils` is available at geo-utils.readthedocs.io.

`geo_utils.geo_utils.`**float2int**(*raster_file_name*, *band_number=1*)

Converts a float number raster to an integer raster (required for converting a raster to a polygon shapefile).

Parameters

- **raster_file_name** (*str*) – Target file name, including directory; must end on `".tif"`.
- **band_number** (*int*) – The raster band number to open (default: `1`).

Returns `"path/to/ew_raster_file.tif"`

Return type str

`geo_utils.geo_utils.`**raster2line**(*raster_file_name*, *out_shp_fn*, *pixel_value*)

Converts a raster to a line shapefile, where `pixel_value` determines line start and end points.

Parameters

- **raster_file_name** (*str*) – of input raster file name, including directory; must end on `".tif"`.
- **out_shp_fn** (*str*) – of target shapefile name, including directory; must end on `".shp"`.
- **pixel_value** – Pixel values to connect.

`geo_utils.geo_utils.`**raster2polygon**(*file_name*, *out_shp_fn*, *band_number=1*, *field_name='values'*)

Converts a raster to a polygon shapefile.

Parameters

- **file_name** (*str*) – Target file name, including directory; must end on ".tif"
- **out_shp_fn** (*str*) – Shapefile name (with directory e.g., "C:/temp/poly.shp")
- **band_number** (*int*) – Raster band number to open (default: 1)
- **field_name** (*str*) – Field name where raster pixel values will be stored (default: "values")
- **add_area** – If True, an "area" field will be added, where the area in the shapefiles unit system is calculated (default: False)

geo_utils.geo_utils.**rasterize**(*in_shp_file_name*, *out_raster_file_name*, *pixel_size=10*, *no_data_value=- 9999*, *rdtype=gdal.GDT_Float32*, *overwrite=True*, *interpolate_gap_pixels=False*, ***kwargs*)

Converts any ESRI shapefile to a raster.

Parameters

- **in_shp_file_name** (*str*) – A shapefile name (with directory e.g., "C:/temp/poly.shp")
- **out_raster_file_name** (*str*) – Target file name, including directory; must end on ".tif"
- **pixel_size** (*float*) – Pixel size as multiple of length units defined in the spatial reference (default: 10)
- **no_data_value** (*int OR float*) – Numeric value for no-data pixels (default: -9999)
- **rdtype** (*gdal.GDALDataType*) – The raster data type (default: gdal.GDT_Float32 (32 bit floating point)
- **overwrite** (*bool*) – Overwrite existing files (default: True)
- **interpolate_gap_pixels** (*bool*) – Fill empty pixels that are not touched by a shapefile element with interpolated values (default: False)

Keyword Arguments

- **field_name** (*str*) – Name of the shapefile's field with values to burn to raster pixel values.
- **radius1** (*float*) – Define the x-radius for interpolating pixels (default: -1, corresponding to infinity). Only applicable with interpolate_gap_pixels.
- **radius2** (*float*) – Define the y-radius for interpolating pixels (default: -1, corresponding to infinity). Only applicable with interpolate_gap_pixels.
- **power** (*float*) – Power of the function for interpolating pixel values (default: 1.0, corresponding to linear).
- **smoothing** (*float*) – Smoothing parameter for interpolating pixel values (default: 0.0).
- **min_points** (*int*) – Minimum number of points to use for interpolation. If the interpolator cannot find at least min_points for a pixel, it assigns a no_data value to that pixel (default: 0).
- **max_points** (*int*) – Maximum number of points to use for interpolation. The interpolator will not use more than max_points closest points to interpolate a pixel value (default: 0).

Hints: More information on pixel value interpolation: * `interpolate_gap_pixels=True` interpolates values at pixels that are not touched by any las point. * The pixel value interpolation uses `gdal_grid` (i.e., its Python bindings through `gdal.Grid()`). * Control the interpolation parameters with the keyword arguments `radius1`, `radius2`, `power`, `max_points`, `min_points`, and `smoothing`..

Returns Creates the GeoTIFF raster defined with `out_raster_file_name` (success: `0`, otherwise `None`).

Return type int

3.5.2 geo_utils raster management (raster_mgmt.py)

`geo_utils.raster_mgmt.`**`clip_raster`**(*polygon*, *in_raster*, *out_raster*)

Clips a raster to a polygon.

Parameters

- **`polygon`** (*`str`*) – A polygon shapefile name, including directory; must end on `".shp"`.
- **`in_raster`** (*`str`*) – Name of the raster to be clipped, including its directory.
- **`out_raster`** (*`str`*) – Name of the target raster, including its directory.

Returns Creates a new, clipped raster defined with `out_raster`.

Return type None

`geo_utils.raster_mgmt.`**`create_raster`**(*file_name*, *raster_array*, *origin=None*, *epsg=4326*, *pixel_width=10.0*, *pixel_height=10.0*, *nan_val=- 9999.0*, *rdtype=gdal.GDT_Float32*, *geo_info=False*)

Converts an `ndarray` (`numpy.array`) to a GeoTIFF raster.

Parameters

- **`file_name`** (*`str`*) – Target file name, including directory; must end on `".tif"`.
- **`raster_array`** (*`ndarray`*) – Values to rasterize.
- **`origin`** (*`tuple`*) – Coordinates (x, y) of the origin.
- **`epsg`** (*`int`*) – EPSG:XXXX projection to use (default: `4326`).
- **`pixel_height`** (*`float OR int`*) – Pixel height as multiple of the base units defined with the EPSG number (default: `10` meters).
- **`pixel_width`** (*`float OR int`*) – Pixel width as multiple of the base units defined with the EPSG number (default: `10` meters).
- **`nan_val`** (`int` or `float`): No-data value to be used in the raster. Replaces non-numeric and `np.nan` in the `ndarray`. (default: `geoconfig.nan_value`) –
- **`rdtype`** – gdal.GDALDataType raster data type (default: gdal.GDT_Float32 (32 bit floating point).
- **`geo_info`** (*`tuple`*) – Defines a `gdal.DataSet.GetGeoTransform` object and supersedes `origin`, `pixel_width`, `pixel_height` (default: `False`).

Returns `0` if successful, otherwise `-1`.

Return type int

`geo_utils.raster_mgmt.`**`open_raster`**(*file_name*, *band_number=1*)

Opens a raster file and accesses its bands.

Parameters

- **file_name** (*str*) – The raster file directory and name.
- **band_number** (*int*) – The Raster band number to open (default: `1`).

Returns A raster dataset a Python object. osgeo.gdal.Band: The defined raster band as Python object.

Return type osgeo.gdal.Dataset

`geo_utils.raster_mgmt.`**`raster2array`**(*file_name*, *band_number=1*)

Extracts an `ndarray` from a raster.

Parameters

- **file_name** (*str*) – Target file name, including directory; must end on `".tif"`.
- **band_number** (*int*) – The raster band number to open (default: `1`).

Returns Indicated raster band, where no-data values are replaced with `np.nan`. GeoTransform: The GeoTransformation used in the original raster.

Return type ndarray

`geo_utils.raster_mgmt.`**`remove_tif`**(*file_name*)

Removes a GeoTIFF and its dependent files (e.g., xml).

Parameters **file_name** (*str*) – Directory (path) and name of a GeoTIFF

Returns Removes the provided `file_name` and all dependencies.

3.5.3 geo_utils shapefile management (shp_mgmt.py)

`geo_utils.shp_mgmt.`**`create_shp`**(*shp_file_dir*, *overwrite=True*, **args*, ***kwargs*)

Creates a new shapefile with an optionally defined geometry type.

Parameters

- **shp_file_dir** (*str*) – of the (relative) shapefile directory (ends on `".shp"`).
- **overwrite** (*bool*) – If `True` (default), existing files are overwritten.
- **layer_name** (*str*) – The layer name to be created. If `None`: no layer will be created.
- **layer_type** (*str*) – Either `"point"`, `"line"`, or `"polygon"` of the `layer_name`. If `None`: no layer will be created.

Returns An `ogr` shapefile

Return type osgeo.ogr.DataSource

`geo_utils.shp_mgmt.`**`get_geom_description`**(*layer*)

Gets the WKB Geometry Type as string from a shapefile layer.

Parameters **layer** (*osgeo.ogr.Layer*) – A shapefile layer.

Returns WKB (binary) geometry type

Return type str

`geo_utils.shp_mgmt.`**`get_geom_simplified`**(*layer*)

Gets a simplified geometry description (either point, line, or polygon) as a function of the WKB Geometry Type of a shapefile layer.

Parameters **layer** (*osgeo.ogr.Layer*) – A shapefile layer.

Returns Either WKT-formatted point, line, or polygon (or unknown if invalid layer).

Return type str

`geo_utils.shp_mgmt.`**`polygon_from_shapepoints`**(*shapepoints*, *polygon*, *alpha=numpy.nan*)
Creates a polygon around a cloud of `shapepoints`.

Parameters

- **shapepoints** (*str*) – Point shapefile name, including its directory.
- **polygon** (*str*) – Target shapefile filename, including its directory.
- **alpha** (*float*) – Coefficient to adjust; the lower it is, the more slim will be the polygon.

Returns Creates the polygon shapefile defined with `polygon`.

Return type None

`geo_utils.shp_mgmt.`**`verify_shp_name`**(*shp_file_name*, *shorten_to=13*)
Ensure that the shapefile name does not exceed 13 characters. Otherwise, the function shortens the `shp_file_name` length to N characters.

Parameters

- **shp_file_name** (*str*) – A shapefile name (with directory e.g., `"C:/temp/poly.shp"`).
- **shorten_to** (*int*) – The number of characters the shapefile name should have (default: `13`).

Returns A shapefile name (including path if provided) with a length of `shorten_to`.

Return type str

3.5.4 geo_utils projection management (srs_mgmt.py)

`geo_utils.srs_mgmt.`**`get_esriwkt`**(*epsg*)
Gets esriwkt-formatted spatial references with epsg code online.

Parameters **epsg** (*int*) – EPSG Authority Code

Returns An esriwkt string (if an error occur, the default epsg=``4326`` is used).

Return type str

Example

```
get_esriwkt(4326)
```

`geo_utils.srs_mgmt.`**`get_srs`**(*dataset*)
Gets the spatial reference of any `gdal.Dataset`.

Parameters **dataset** (*gdal.Dataset*) – A shapefile or raster.

Returns A spatial reference object.

Return type osr.SpatialReference

`geo_utils.srs_mgmt.`**`get_wkt`**(*epsg*, *wkt_format='esriwkt'*)
Gets WKT-formatted projection information for an epsg code using the `osr` library.

Parameters

- **epsg** (*int*) – epsg Authority code
- **wkt_format** (*str*) – of wkt format (default is esriwkt for shapefile projections)

Returns WKT (if error: returns default corresponding to `epsg=4326`).

Return type str

`geo_utils.srs_mgmt.`**`make_prj`**(*shp_file_name*, *epsg*)

Generates a projection file for a shapefile.

Parameters

- **shp_file_name** (*str*) – of a shapefile name (with directory e.g., `"C:/temp/poly.shp"`).
- **epsg** (*int*) – EPSG Authority Code

Returns Creates a projection file (`.prj`) in the same directory and with the same name of `shp_file_name`.

`geo_utils.srs_mgmt.`**`reproject`**(*source_dataset*, *new_projection_dataset*)

Re-projects a dataset (raster or shapefile) onto the spatial reference system of a (shapefile or raster) layer.

Parameters

- **source_dataset** (*gdal.Dataset*) – Shapefile or raster.
- **new_projection_dataset** (*gdal.Dataset*) – Shapefile or raster with new projection info.

Returns

- If the source is a raster, the function creates a GeoTIFF in same directory as `source_dataset` with a `"_reprojected"` suffix in the file name.
- If the source is a shapefile, the function creates a shapefile in same directory as `source_dataset` with a `"_reprojected"` suffix in the file name.

`geo_utils.srs_mgmt.`**`reproject_raster`**(*source_dataset*, *source_srs*, *target_srs*)

Re-projects a raster dataset. This function is called by the `reproject` function.

Parameters

- **source_dataset** (*osgeo.ogr.DataSource*) – Instantiates with an `ogr.Open(SHP-FILE)`.
- **source_srs** (*osgeo.osr.SpatialReference*) – Instantiates with `get_srs(source_dataset)`
- **target_srs** (*osgeo.osr.SpatialReference*) – Instantiates with `get_srs(DATASET-WITH-TARGET-PROJECTION)`.

Returns Creates a new GeoTIFF raster in the same directory where `source_dataset` lives.

`geo_utils.srs_mgmt.`**`reproject_shapefile`**(*source_dataset*, *source_layer*, *source_srs*, *target_srs*)

Re-projects a shapefile dataset. This function is called by the `reproject` function.

Parameters

- **source_dataset** (*osgeo.ogr.DataSource*) – Instantiates with `ogr.Open(SHP-FILE)`.

- **source_layer** (*osgeo.ogr.Layer*) – Instantiates with `source_dataset.GetLayer()`.
- **source_srs** (*osgeo.osr.SpatialReference*) – Instantiates with `get_srs(source_dataset)`.
- **target_srs** (*osgeo.osr.SpatialReference*) – Instantiates with `get_srs(DATASET-WITH-TARGET-PROJECTION)`.

Returns Creates a new shapefile in the same directory where `source_dataset` lives.

3.5.5 geo_utils dataset Conversion (dataset_mgmt.py)

`geo_utils.dataset_mgmt.`**`coords2offset`**(*geo_transform*, *x_coord*, *y_coord*)
Returns x-y pixel offset (inverse of the `offset2coords` function).

Parameters

- **geo_transform** – osgeo.gdal.Dataset.GetGeoTransform() object
- **x_coord** (*float*) – x-coordinate
- **y_coord** (*float*) – y-coordinate

Returns Number of pixels `(offset_x, offset_y)`, both `int`.

Return type tuple

`geo_utils.dataset_mgmt.`**`get_layer`**(*dataset*, *band_number=1*)
Gets a `layer=band` (`RasterDataSet`) or `layer=ogr.Dataset.Layer` of any dataset.

Parameters

- **dataset** (`osgeo.gdal.Dataset` or `osgeo.ogr.DataSource`) – Either a raster or a shapefile.
- **band_number** (*int*) – Only use with rasters to define a band number to open (default=``1``).

Returns `{GEO-TYPE: if raster: raster_band, if vector: GetLayer(), else: None}`

Return type dict

`geo_utils.dataset_mgmt.`**`offset2coords`**(*geo_transform*, *offset_x*, *offset_y*)
Returns x-y coordinates from pixel offset (inverse of `coords2offset` function).

Parameters

- **geo_transform** (*osgeo.gdal.Dataset.GetGeoTransform*) – The geo transformation to use.
- **offset_x** (*int*) – x number of pixels.
- **offset_y** (*int*) – y number of pixels.

Returns Two `float` numbers of x-y-coordinates `(x_coord, y_coord)`.

Return type tuple

`geo_utils.dataset_mgmt.`**`verify_dataset`**(*dataset*)
Verifies if a dataset contains raster or vector data.

Parameters **dataset** (`osgeo.gdal.Dataset` or `osgeo.ogr.DataSource`) – Dataset to verify.

Returns Either "mixed", "raster", or "vector".

Return type string

CHAPTER

FOUR

TROUBLESHOOTING

4.1 Memory errors

MemoryError

Cause: *las* file may have size of several GiB, which may quickly cause a `MemoryError` (e.g., `MemoryError: Unable to allocate 9.1 GiB for an array with shape ...`). In particular the *Linux* kernel will not attempt to run actions that exceed the commit-able memory.

Solution: Enable memory over-committing:

- **Check the current over-commit mode in *Terminal*:** `cat /proc/sys/vm/overcommit_memory`
- If `0` is the answer, the system calculates array dimensions and the required memory (e.g., an array with dimensions `(266838515, 12, 49)` requires a memory of `266838515 * 12 *49 / 1024.0**3` = `146` GiB, which is unlikely to fit in the memory).
- **To enable over-committing, set the commit mode to `1`:** `echo 1 | sudo tee /proc/sys/vm/overcommit_memory`

CHAPTER

FIVE

CONTRIBUTE (DEVELOPERS)

5.1 How to document hylas

This package uses *Sphinx* readthedocs and the documentation regenerates automatically after pushing changes to the repositories `main` branch.

To set styles, configure or add extensions to the documentation use `ROOT/.readthedocs.yml` and `ROOT/docs/conf.py`.

Functions and classes are automatically parsed for docstrings and implemented in the documentation. `hylas` docs use google style docstring formats - please familiarize with the style format and strictly apply in all commits.

To modify this documentation file, edit `ROOT/docs/index.rst` (uses reStructuredText format).

In the class or function docstrings use the following section headers:

- `Args (alias of Parameters)`
- `Arguments (alias of Parameters)`
- `Attention`
- `Attributes`
- `Caution`
- `Danger`
- `Error`
- `Example`
- `Examples`
- `Hint`
- `Important`
- `Keyword Args (alias of Keyword Arguments)`
- `Keyword Arguments`
- `Methods`
- `Note`
- `Notes`
- `Other Parameters`
- `Parameters`
- `Return (alias of Returns)`

- `Returns`
- `Raise (alias of Raises)`
- `Raises`
- `References`
- `See Also`
- `Tip`
- `Todo`
- `Warning`
- `Warnings (alias of Warning)`
- `Warn (alias of Warns)`
- `Warns`
- `Yield (alias of Yields)`
- `Yields`

For local builds of the documentation, the following packages are required:

```
sudo apt-get install build-essential
sudo apt-get install python-dev python-pip python-setuptools
sudo apt-get install libxml2-dev libxslt1-dev zlib1g-dev
apt-cache search libffi
sudo apt-get install -y libffi-dev
sudo apt-get install python3-dev default-libmysqlclient-dev
sudo apt-get install python3-dev
sudo apt-get install redis-server
```

To generate a local html version of the `hylas` documentation, `cd` into the `docs` directory and type:

```
make html
```

Learn more about *Sphinx* documentation and the automatic generation of *Python* code docs through docstrings in the tutorial provided at github.com/sschwindt/docs-with-sphinx.

CHAPTER

SIX

INDICES AND TABLES

- :ref:`genindex`
- :ref:`modindex`
- :ref:`search`

CHAPTER

SEVEN

DISCLAIMER AND LICENSE

7.1 Disclaimer (general)

No warranty is expressed or implied regarding the usefulness or completeness of the information provided for *hylas* and its documentation. References to commercial products do not imply endorsement by the Authors of *hylas*. The concepts, materials, and methods used in the codes and described in the docs are for informational purposes only. The Authors have made substantial effort to ensure the accuracy of the code and the docs and the Authors shall not be held liable, nor their employers or funding sponsors, for calculations and/or decisions made on the basis of application of *hylas*. The information is provided "as is" and anyone who chooses to use the information is responsible for her or his own choices as to what to do with the code, docs, and data and the individual is responsible for the results that follow from their decisions.

7.2 BSD 3-Clause License

Copyright (c) 2020, Sebastian Schwindt and all other the Authors of *hylas*. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

PYTHON MODULE INDEX

INDEX

P

R

S

V

W