Telluric Documentation

Release v0.14.4

Juan Luis Cano, Slava Kerner, Lucio Torre

Apr 18, 2023

CONTENTS:

telluric is a Python library to manage vector and raster geospatial data in an interactive and easy way.

The source code and issue tracker are hosted on GitHub, and all contributions and feedback are more than welcome. There is a public chat for users and developers too.

CHAPTER

ONE

INSTALLATION

You can install telluric using pip:

```
pip install telluric[vis]
```

telluric is a pure Python library, and therefore should work on Linux, OS X and Windows provided that you can install its dependencies. If you find any problem, please open an issue and we will take care of it.

> **Warning:** It is recommended that you **never ever use sudo** with pip because you might seriously break your system. Use venv, Pipenv, pyenv or conda to create an isolated development environment instead.

1.1 User Guide

1.1.1 Geometries on a map: `GeoVector`

[1]:
```
import telluric as tl
from telluric.constants import WGS84_CRS, WEB_MERCATOR_CRS
```

The simplest geometrical element in telluric is the GeoVector: it represents a shape in some coordinate reference system (CRS). The easiest way to create one is to use the `GeoVector.from_bounds` method:

[2]:
```
gv1 = tl.GeoVector.from_bounds(
    xmin=0, ymin=40, xmax=1, ymax=41, crs=WGS84_CRS
)
print(gv1)
```

```
GeoVector(shape=POLYGON ((0 40, 0 41, 1 41, 1 40, 0 40)), crs=CRS({'init': 'epsg:4326'}))
```

If we print the object, we see its two defining elements: a shape (actually a shapely `BaseGeometry` object) and a CRS (in this case WGS84 or http://epsg.io/4326). Rather than reading a dull representation, we can directly visualize it in the notebook:

[3]:
```
gv1
```

```
/home/juanlu/Satellogic/telluric/telluric/plotting.py:141: UserWarning: Plotting a␣
↪limited representation of the data, use the .plot() method for further customization
  "Plotting a limited representation of the data, use the .plot() method for further␣
↪customization")
```

[3]:

You can ignore the warning for the moment. Advanced plotting techniques are not yet covered in this User Guide.

As you can see, we have an interactive Web Mercator map where we can display our shape. We can create more complex objects using the Shapely library:

[4]:
```
from shapely.geometry import Polygon

gv2 = tl.GeoVector(
    Polygon([(0, 40), (1, 40.1), (1, 41), (-0.5, 40.5), (0, 40)]),
    WGS84_CRS
)
print(gv2)
```

```
GeoVector(shape=POLYGON ((0 40, 1 40.1, 1 41, -0.5 40.5, 0 40)), crs=CRS({'init': 'epsg:
↪4326'}))
```

And we can access any property of the underlying geometry using the same attribute name:

[5]:
```
print(gv1.centroid)
```

```
GeoVector(shape=POINT (0.5 40.5), crs=CRS({'init': 'epsg:4326'}))
```

[6]:
```
gv1.area  # Real area in square meters
```

[6]:
```
9422706289.175217
```

[7]:
```
gv1.is_valid
```

[7]:
```
True
```

[8]:
```
gv1.within(gv2)
```

[8]:
```
False
```

[9]:
```
gv1.difference(gv2)
```

```
/home/juanlu/Satellogic/telluric/telluric/plotting.py:141: UserWarning: Plotting a␣
↪limited representation of the data, use the .plot() method for further customization
  "Plotting a limited representation of the data, use the .plot() method for further␣
↪customization")
```

[9]:

1.1.2 Geometries with attributes: `GeoFeature` and `FeatureCollection`

The next object in the telluric hierarchy is the GeoFeature: a combination of a `GeoVector` + some attributes. These attributes can represent land use, types of buildings, and so forth.

[10]:
```
gf1 = tl.GeoFeature(
    gv1,
    {'name': 'One feature'}
)
```

(continues on next page)

(continued from previous page)

```
gf2 = tl.GeoFeature(
    gv2,
    {'name': 'Another feature'}
)
print(gf1)
print(gf2)
```

```
GeoFeature(Polygon, {'name': 'One feature'})
GeoFeature(Polygon, {'name': 'Another feature'})
```

But the most interesting thing is to combine these features into a FeatureCollection. A `FeatureCollection` is essentially a sequence of features, with some additional methods:

[11]:
```
fc = tl.FeatureCollection([gf1, gf2])
fc
```

```
/home/juanlu/Satellogic/telluric/telluric/plotting.py:141: UserWarning: Plotting a␣
↪limited representation of the data, use the .plot() method for further customization
  "Plotting a limited representation of the data, use the .plot() method for further␣
↪customization")
```

[11]:
```
<telluric.collections.FeatureCollection at 0x7f283ea41f60>
```

[12]:
```
print(fc.convex_hull)
```

```
GeoVector(shape=POLYGON ((0 40, -0.5 40.5, 0 41, 1 41, 1 40, 0 40)), crs=CRS({'init':
↪'epsg:4326'}))
```

[13]:
```
print(fc.envelope)
```

```
GeoVector(shape=POLYGON ((-0.5 40, 1 40, 1 41, -0.5 41, -0.5 40)), crs=CRS({'init':
↪'epsg:4326'}))
```

1.1.3 Input and Output

Apart from all the previous geospatial operations, we can also save these `FeatureCollection` objects to disk, for example using the GeoJSON or ESRI Shapefile formats:

[14]:
```
fc.save("test_fc.shp")
```

[15]:
```
!ls test_fc*
```

```
test_fc.cpg  test_fc.dbf  test_fc.json       test_fc.prj  test_fc.shp  test_fc.shx
```

[16]:
```
fc.save("test_fc.json")
```

[17]:
```
!python -m json.tool < test_fc.json | head -n28
```

```
{
    "type": "FeatureCollection",
    "crs": {
        "type": "name",
        "properties": {
```

(continues on next page)

(continued from previous page)

```
            "name": "urn:ogc:def:crs:OGC:1.3:CRS84"
        }
    },
    "features": [
        {
            "type": "Feature",
            "properties": {
                "name": "One feature",
                "highlight": {},
                "style": {}
            },
            "geometry": {
                "type": "Polygon",
                "coordinates": [
                    [
                        [
                            0.0,
                            40.0
                        ],
                        [
                            0.0,
                            41.0
                        ],
```

To retrieve this data from disk again, we can use another object, `FileCollection`, which behaves in the same way as a `FeatureCollection` but does some smart optimizations so the files are not read completely into memory:

[18]:
```
print(list(tl.FileCollection.open("test_fc.shp")))
```

```
[GeoFeature(Polygon, {'name': 'One feature', 'highlight': '{}', 'style': '{}'}),␣
↪GeoFeature(Polygon, {'name': 'Another feature', 'highlight': '{}', 'style': '{}'})]
```

1.1.4 Raster data: `GeoRaster2`

After reviewing how to read, manipulate and write vector data, we can use GeoRaster2 to do the same thing with raster data. `GeoRaster2` will read the raster lazily so we only retrieve the information that we need.

[19]:
```
# This will only save the URL in memory
rs = tl.GeoRaster2.open(
    "https://github.com/mapbox/rasterio/raw/master/tests/data/rgb_deflate.tif"
)

# These calls will fecth some GeoTIFF metadata
# without reading the whole image
print(rs.crs)
print(rs.footprint())
print(rs.band_names)
```

```
CRS({'init': 'epsg:32618'})
GeoVector(shape=POLYGON ((101984.9999999127 2826915, 339314.9999997905 2826915, 339314.
↪9999998778 2611485, 101985.0000002096 2611485, 101984.9999999127 2826915)), crs=CRS({
```

(continues on next page)

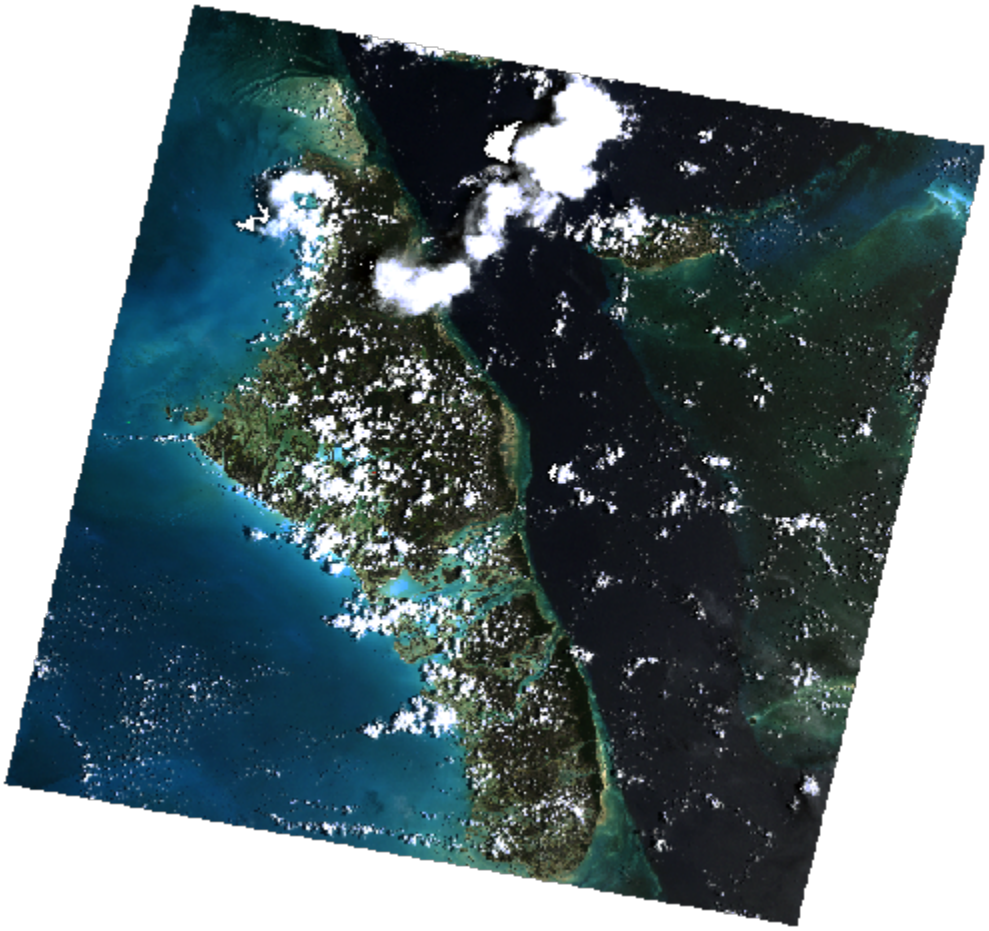
(continued from previous page)

```
↪'init': 'epsg:32618'}))
[0, 1, 2]
```

GeoRaster2 also displays itself automatically:

[20]:
```
rs
```

[20]:



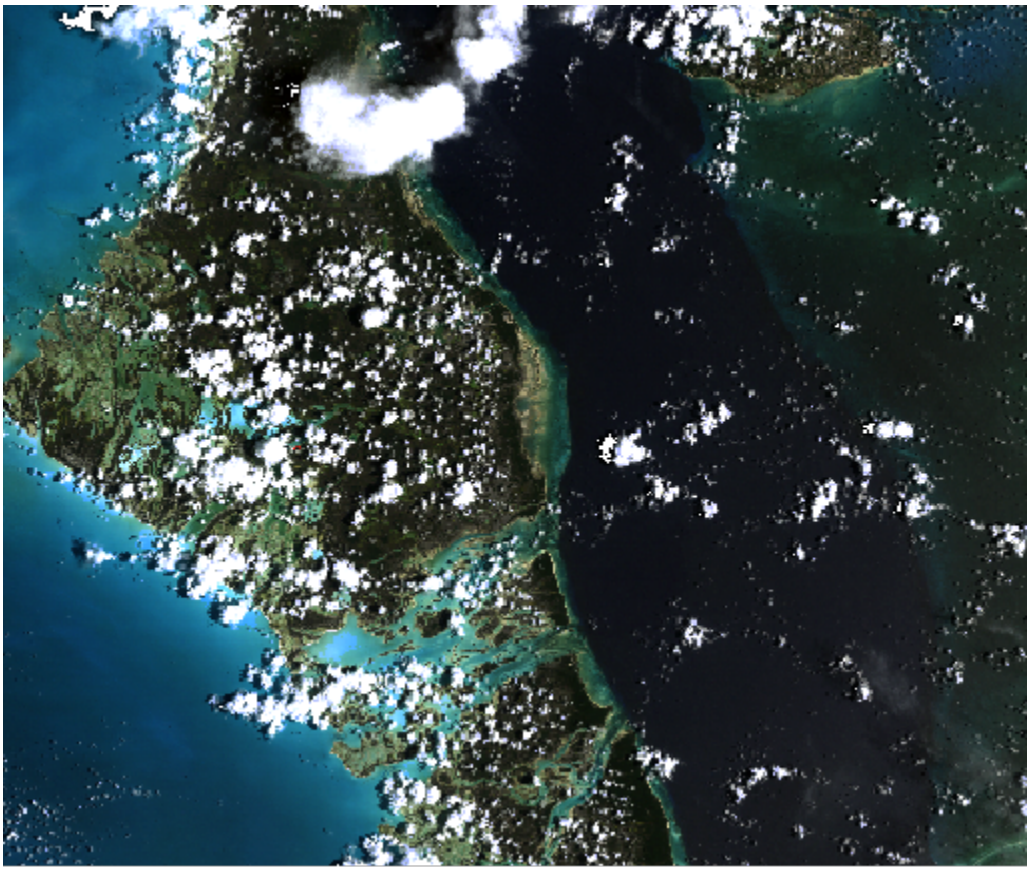
We can slice it like an array, or cropping some parts to discard others:

[21]:
```
rs.shape
```

[21]:
```
(3, 718, 791)
```

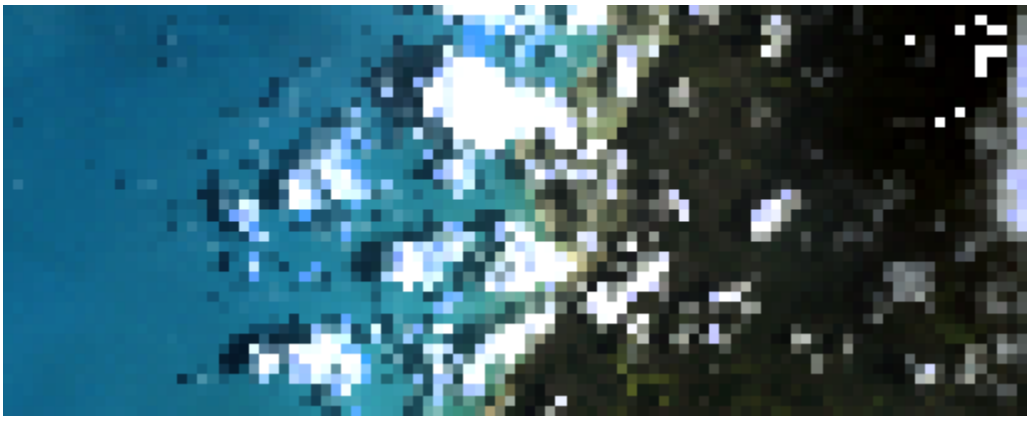
[22]:
```
rs.crop(rs.footprint().buffer(-50000))
```

[22]:



[23]:
```
rs[200:300, 200:240]
```

[23]:



And save again to GeoTIFF format using a variety of options:

[24]:
```
rs[200:300, 200:240].save("test_raster.tif")
```

1.1.5 Conclusion

There are many things not covered in this User Guide. The documentation of telluric is a work in progress, so we encourage you to read the full API reference and even contribute to the package!

1.2 API Reference

1.2.1 telluric.constants module

Useful constants.

telluric.constants.**DEFAULT_CRS = CRS.from_epsg(4326)**

Default CRS, set to *WGS84_CRS*.

telluric.constants.**EQUAL_AREA_CRS = CRS.from_wkt('PROJCS["unknown",GEOGCS["unknown",DATUM["WGS_1984",SPHEROID["WGS 84",6378137,298.257223563,AUTHORITY["EPSG","7030"]],AUTHORITY["EPSG","6326"]],PRIMEM["Greenwich",0,AUTHORITY["EPSG","8901"]],UNIT["degree",0.0174532925199433,AUTHORITY["EPSG","9122"]]],PROJECTION["Eckert_IV"],PARAMETER["central_meridian",0],PARAMETER["false_easting",0],PARAMETER["false_northing",0],UNIT["metre",1,AUTHORITY["EPSG","9001"]],AXIS["Easting",EAST],AXIS["Northing",NORTH]]')**

Eckert IV CRS.

telluric.constants.**WEB_MERCATOR_CRS = CRS.from_epsg(3857)**

Web Mercator CRS.

telluric.constants.**WGS84_CRS = CRS.from_epsg(4326)**

WGS84 CRS.

1.2.2 telluric.vectors module

class telluric.vectors.**GeoVector**(*shape*, *crs=CRS.from_epsg(4326)*, *safe=True*)

Geometric element with an associated CRS.

This class has also all the properties and methods of `shapely.geometry.BaseGeometry`.

__init__(*shape*, *crs=CRS.from_epsg(4326)*, *safe=True*)

Initialize GeoVector.

Parameters

- **shape** (*shapely.geometry.BaseGeometry*) – Geometry.
- **crs** (*CRS (optional)*) – Coordinate Reference System, default to *telluric.constants.DEFAULT_CRS*.
- **safe** (*bool, optional*) – Check method arguments validity (does nothing so far) if False, default to True

almost_equals(*other*, *decimal=6*)

invariant to crs.

classmethod cascaded_union(*vectors: list*, *dst_crs: CRS*, *prevalidate: bool = False*) → *GeoVector*

Generate a GeoVector from the cascade union of the impute vectors.

copy()

make a copy of the GeoVector.

equals_exact(*other*, *tolerance*)

invariant to crs.

classmethod from_bounds(*xmin*, *ymin*, *xmax*, *ymax*, *crs=CRS.from_epsg(4326)*)

Creates GeoVector object from bounds.

Parameters

- **xmin** (*float*) – Bounds of the GeoVector. Also (east, south, north, west).
- **ymin** (*float*) – Bounds of the GeoVector. Also (east, south, north, west).
- **xmax** (*float*) – Bounds of the GeoVector. Also (east, south, north, west).
- **ymax** (*float*) – Bounds of the GeoVector. Also (east, south, north, west).
- **crs** (*CRS, dict*) – Projection, default to *telluric.constants.DEFAULT_CRS*.

Examples

```
>>> from telluric import GeoVector
>>> GeoVector.from_bounds(xmin=0, ymin=0, xmax=1, ymax=1)
GeoVector(shape=POLYGON ((0 0, 0 1, 1 1, 1 0, 0 0)), crs=CRS({'init': 'epsg:4326
↪'}))
>>> GeoVector.from_bounds(xmin=0, xmax=1, ymin=0, ymax=1)
GeoVector(shape=POLYGON ((0 0, 0 1, 1 1, 1 0, 0 0)), crs=CRS({'init': 'epsg:4326
↪'}))
```

classmethod from_geojson(*filename*)

Load vector from geojson.

classmethod from_record(*record*, *crs*)

Load vector from record.

classmethod from_xyz(*x*, *y*, *z*)

Creates GeoVector from Mercator slippy map values.

get_bounding_box(*crs*)

Gets bounding box as GeoVector in a specified CRS.

get_shape(*crs*)

Gets the underlying Shapely shape in a specified CRS.

This method deliberately does not have a default crs=self.crs to force the user to specify it.

polygonize(*width*, *cap_style_line=2*, *cap_style_point=1*)

Turns line or point into a buffered polygon.

tiles(*zooms*, *truncate=False*)

Iterator over the tiles intersecting the bounding box of the vector

Parameters

- **zooms** (*int or sequence of int*) – One or more zoom levels.
- **truncate** (*bool, optional*) – Whether or not to truncate inputs to web mercator limits.

Yields
mercantile.Tile object (*namedtuple* with x, y, z)

`to_geojson`(*filename*)

Save vector as geojson.

`telluric.vectors.generate_tile_coordinates`(*roi: GeoVector*, *num_tiles: Tuple[int, int]*) → Iterator[*GeoVector*]

Yields N x M rectangular tiles for a region of interest.

Parameters

- **roi** (GeoVector) – Region of interest
- **num_tiles** (*tuple*) – Tuple (horizontal_tiles, vertical_tiles)

Yields
~telluric.vectors.GeoVector

`telluric.vectors.generate_tile_coordinates_from_pixels`(*roi*, *scale*, *size*)

Yields N x M rectangular tiles for a region of interest.

Parameters

- **roi** (GeoVector) – Region of interest
- **scale** (*float*) – Scale factor (think of it as pixel resolution)
- **size** (*tuple*) – Pixel size in (width, height) to be multiplied by the scale factor

Yields
~telluric.vectors.GeoVector

`telluric.vectors.get_dimension`(*geometry*)

Gets the dimension of a Fiona-like geometry element.

1.2.3 telluric.features module

`class telluric.features.GeoFeature`(*geovector*, *properties*, *assets=None*)

GeoFeature object.

`__init__`(*geovector*, *properties*, *assets=None*)

Initialize a GeoFeature object.

Parameters

- **geovector** (GeoVector) – Geometry.
- **properties** (*dict*) – Properties.

`copy_with`(*geometry=None*, *properties=None*, *assets=None*)

Generate a new GeoFeature with different geometry or preperties.

`classmethod from_raster`(*raster*, *properties*, *product='visual'*)

Initialize a GeoFeature object with a GeoRaster

Parameters

- **raster** (*GeoRaster*) – the raster in the feature
- **properties** (*dict*) – Properties.

- **product** (*str*) – product associated to the raster

classmethod from_record(*record*, *crs*, *schema=None*)

Create GeoFeature from a record.

get_shape(*crs*)

Gets the underlying Shapely shape in a specified CRS.

property has_raster

True if any of the assets is type 'raster'.

raster(*name=None*, ***creteria*)

Generates a GeoRaster2 object based on the asset name(key) or a creteria(protety name and value).

exception telluric.features.**GeoFeatureError**

telluric.features.**serialize_properties**(*properties*)

Serialize properties.

Parameters

properties (*dict*) – Properties to serialize.

telluric.features.**transform_properties**(*properties*, *schema*)

Transform properties types according to a schema.

Parameters

- **properties** (*dict*) – Properties to transform.
- **schema** (*dict*) – Fiona schema containing the types.

1.2.4 telluric.collections module

class telluric.collections.**BaseCollection**

apply(***kwargs*)

Return a new FeatureCollection with the results of applying the statements in the arguments to each element.

dissolve(*by: Optional[str] = None*, *aggfunc: Optional[Callable] = None*) → *FeatureCollection*

Dissolve geometries and rasters within *groupby*.

filter(*intersects*)

Filter results that intersect a given GeoFeature or Vector.

get_values(*key*)

Get all values of a certain property.

groupby(*by: Union[str, Callable[[GeoFeature], str]]*) → _CollectionGroupBy

Groups collection using a value of a property.

Parameters

by (*str or callable*) – If string, name of the property by which to group. If callable, should receive a GeoFeature and return the category.

Return type

_CollectionGroupBy

property is_empty

True if all features are empty.

map(*map_function*)

Return a new FeatureCollection with the results of applying *map_function* to each element.

rasterize(*dest_resolution*, ***, *polygonize_width=0*, *crs=CRS.from_epsg(3857)*, *fill_value=None*, *bounds=None*, *dtype=None*, ***polygonize_kwargs*)

Binarize a FeatureCollection and produce a raster with the target resolution.

Parameters

- **dest_resolution** (*float*) – Resolution in units of the CRS.
- **polygonize_width** (*int, optional*) – Width for the polygonized features (lines and points) in pixels, default to 0 (they won't appear).
- **crs** (*CRS, dict (optional)*) – Coordinate system, default to *telluric.constants.WEB_MERCATOR_CRS*.
- **fill_value** (*float or function, optional*) – Value that represents data, default to None (will default to `telluric.rasterization.FILL_VALUE`. If given a function, it must accept a single *GeoFeature* and return a numeric value.
- **nodata_value** (*float, optional*) – Nodata value, default to None (will default to `telluric.rasterization.NODATA_VALUE`.
- **bounds** (*GeoVector, optional*) – Optional bounds for the target image, default to None (will use the FeatureCollection convex hull).
- **dtype** (*numpy.dtype, optional*) – dtype of the result, required only if fill_value is a function.
- **polygonize_kwargs** (*dict*) – Extra parameters to the polygonize function.

save(*filename*, *driver=None*, *schema=None*)

Saves collection to file.

sort(*by*, *desc=False*)

Sorts by given property or function, ascending or descending order.

Parameters

- **by** (*str or callable*) – If string, property by which to sort. If callable, it should receive a GeoFeature a return a value by which to sort.
- **desc** (*bool, optional*) – Descending sort, default to False (ascending).

class telluric.collections.**FeatureCollection**(*results*, *schema=None*)

__init__(*results*, *schema=None*)

Initialize FeatureCollection object.

Parameters

results (*iterable*) – Iterable of *GeoFeature* objects.

classmethod from_georasters(*georasters*)

Builds new FeatureCollection from a sequence of *GeoRaster2* objects.

classmethod from_geovectors(*geovectors*)

Builds new FeatureCollection from a sequence of *GeoVector* objects.

validate()

if schema exists we run shape file validation code of fiona by trying to save to in MemoryFile

exception telluric.collections.**FeatureCollectionIOError**

class telluric.collections.**FileCollection**(*filename*, *crs*, *schema*, *length*)

FileCollection object.

__init__(*filename*, *crs*, *schema*, *length*)

Initialize a FileCollection object.

Use the *open()* method instead.

classmethod open(*filename*, *crs=None*)

Creates a FileCollection from a file in disk.

Parameters

- **filename** (*str*) – Path of the file to read.
- **crs** (*CRS*) – overrides the crs of the collection, this funtion will not reprojects

telluric.collections.**dissolve**(*collection: BaseCollection*, *aggfunc: Optional[Callable[[list], Any]] = None*) → *GeoFeature*

Dissolves features contained in a FeatureCollection and applies an aggregation function to its properties.

1.2.5 telluric.georaster module

class telluric.georaster.**GeoMultiRaster**(*rasters*)

__init__(*rasters*)

Create a GeoRaster object

Parameters

- **filename** – optional path/url to raster file for lazy loading
- **image** – optional supported: np.ma.array, np.array, TODO: PIL image
- **affine** – affine.Affine, or 9 numbers: [step_x, 0, origin_x, 0, step_y, origin_y, 0, 0, 1]
- **crs** – wkt/epsg code, e.g. {'init': 'epsg:32620'}
- **band_names** – e.g. ['red', 'blue'] or 'red'
- **shape** – raster image shape, optional
- **nodata** – if provided image is array (not masked array), treat pixels with value=nodata as nodata
- **rpcs** – rasterio.rpc.RPC object or dictionary with RPCs values with capital str keys and str values, e.g: {"HEIGHT_OFF":"1.0", "LINE_DEN_COEFF":"0 6.5 0.1 ...",...} or dictionary with RPCs values with capital str keys and float values, e.g: {"HEIGHT_OFF":1.0, "LINE_DEN_COEFF":[0, 6.5, 0.1 ...]",...}
- **temporary** – True means that file referenced by filename is temporary and will be removed by destructor, default False

copy()

Return a copy of this GeoRaster with no modifications.

Can be use to create a Mutable copy of the GeoRaster

class telluric.georaster.**GeoRaster2**(*image=None*, *affine=None*, *crs=None*, *filename=None*, *band_names=None*, *nodata=None*, *shape=None*, *footprint=None*, *rpcs=None*, *temporary=False*)

Represents multiband georeferenced image, supporting nodata pixels. The name "GeoRaster2" is temporary.

conventions:

- .array is np.masked_array, mask=True on nodata pixels.
- .array is [band, y, x]
- .affine is affine.Affine
- .crs is rasterio.crs.CRS
- .band_names is list of strings, order corresponding to order in .array

__init__(*image=None*, *affine=None*, *crs=None*, *filename=None*, *band_names=None*, *nodata=None*, *shape=None*, *footprint=None*, *rpcs=None*, *temporary=False*)

Create a GeoRaster object

Parameters

- **filename** – optional path/url to raster file for lazy loading
- **image** – optional supported: np.ma.array, np.array, TODO: PIL image
- **affine** – affine.Affine, or 9 numbers: [step_x, 0, origin_x, 0, step_y, origin_y, 0, 0, 1]
- **crs** – wkt/epsg code, e.g. {'init': 'epsg:32620'}
- **band_names** – e.g. ['red', 'blue'] or 'red'
- **shape** – raster image shape, optional
- **nodata** – if provided image is array (not masked array), treat pixels with value=nodata as nodata
- **rpcs** – rasterio.rpc.RPC object or dictionary with RPCs values with capital str keys and str values, e.g: {"HEIGHT_OFF":"1.0", "LINE_DEN_COEFF":"0 6.5 0.1 ...",...} or dictionary with RPCs values with capital str keys and float values, e.g: {"HEIGHT_OFF":1.0, "LINE_DEN_COEFF":[0, 6.5, 0.1 ...]",...}
- **temporary** – True means that file referenced by filename is temporary and will be removed by destructor, default False

add_raster(*other*, *merge_strategy*, *resampling*)

Return merge of 2 rasters, in geography of the first one.

merge_strategy - for pixels with values in both rasters.

property affine

Raster affine.

apply_transform(*transformation*, *resampling*)

Apply affine transformation on image & georeferencing.

as specific cases, implement 'resize', 'rotate', 'translate'

astype(*dst_type*, *in_range='dtype'*, *out_range='dtype'*, *clip_negative=False*)

Returns copy of the raster, converted to desired type Supported types: uint8, uint16, uint32, int8, int16, int32, float16, float32, float64

Parameters

- **dst_type** – desired type
- **in_range** – str or 2-tuple, default 'dtype': 'image': use image min/max as the intensity range, 'dtype': use min/max of the image's dtype as the intensity range, 2-tuple: use explicit min/max intensities, it is possible to use 'min' or 'max' as tuple values - in this case they will be replaced by min or max intensity of image respectively
- **out_range** – str or 2-tuple, default 'dtype': 'dtype': use min/max of the image's dtype as the intensity range, 2-tuple: use explicit min/max intensities
- **clip_negative** – boolean, if *True* - clip the negative range, default False

Returns
numpy array of values

attributes(*url*)

Without opening image, return size/bitness/bands/geography/....

property band_names

Raster affine.

block_shape(*band=None*)

Raster single band block shape.

property blockshapes

Raster all bands block shape.

bounds()

Return image rectangle in pixels, as shapely.Polygon.

center()

Return footprint center in world coordinates, as GeoVector.

chunks(*shape=256*, *pad=False*)

This method returns GeoRaster chunks out of the original raster.

The chunck is evaluated only when fetched from the iterator. Useful when you want to iterate over a big rasters.

Parameters

- **shape** (*int or tuple, optional*) – The shape of the chunk. Default: 256.
- **pad** (*bool, optional*) – When set to True all rasters will have the same shape, when False the edge rasters will have a shape less than the requested shape, according to what the raster actually had. Defaults to False.

Returns
out – The iterator that has the raster and the offsets in it.

Return type
RasterChunk

colorize(*colormap*, *band_name=None*, *vmin=None*, *vmax=None*)

Apply a colormap on a selected band.

colormap list: https://matplotlib.org/examples/color/colormaps_reference.html

Parameters

- **colormap** (*str*) –
- **https** (*Colormap name from this list*) –

- **band_name** (*str, optional*) –
- **colorize** (*Name of band to*) –
- **used** (*if None actual raster values will be*) –
- **vmin** (*int, optional*) –
- **vmax** (*int, optional*) –
- **values** (*minimum and maximum range for normalizing array*) –
- **used** –

Return type
GeoRaster2

copy(*mutable=False*)

Return a copy of this GeoRaster with no modifications.

Can be use to create a Mutable copy of the GeoRaster

copy_with(*mutable=None*, ***kwargs*)

Get a copy of this GeoRaster with some attributes changed. NOTE: image is shallow-copied!

corner(*corner*)

Return footprint origin in world coordinates, as GeoVector.

corners()

Return footprint corners, as {corner_type -> GeoVector}.

crop(*vector*, *resolution=None*, *masked=None*, *bands=None*, *resampling=Resampling.cubic*)

crops raster outside vector (convex hull) :param vector: GeoVector, GeoFeature, FeatureCollection :param resolution: output resolution, None for full resolution :param resampling: reprojection resampling method, default *cubic*

Returns
GeoRaster

property crs

Raster crs.

deepcopy_with(*mutable=None*, ***kwargs*)

Get a copy of this GeoRaster with some attributes changed. NOTE: image is shallow-copied!

classmethod from_bytes(*image_bytes*, *affine*, *crs*, *band_names=None*)

Create GeoRaster from image BytesIo object.

Parameters

- **image_bytes** – io.BytesIO object
- **affine** – rasters affine
- **crs** – rasters crs
- **band_names** – e.g. [‘red’, ‘blue’] or ‘red’

classmethod from_rasters(*rasters*, *relative_to_vrt=True*, *destination_file=None*, *nodata=None*, *mask_band=None*)

Create georaster out of a list of rasters.

classmethod from_tiles(*tiles*)

Compose raster from tiles. return GeoRaster.

classmethod from_wms(*filename*, *vector*, *resolution*, *destination_file=None*)

Create georaster from the web service definition file.

get(*point*)

Get the pixel values at the requested point.

Parameters
point – A GeoVector(POINT) with the coordinates of the values to get

Returns
numpy array of values

get_tile(*x_tile*, *y_tile*, *zoom*, *bands=None*, *masked=None*, *resampling=Resampling.cubic*)

Convert mercator tile to raster window.

Parameters

- **x_tile** – x coordinate of tile
- **y_tile** – y coordinate of tile
- **zoom** – zoom level
- **bands** – list of indices of requested bands, default None which returns all bands
- **resampling** – reprojection resampling method, default *cubic*

Returns
GeoRaster2 of tile in WEB_MERCATOR_CRS

You can use TELLURIC_GET_TILE_BUFFER env variable to control the number of pixels surrounding the vector you should fetch when using this method on a raster that is not in WEB_MERCATOR_CRS default to 10

get_window(*window*, *bands=None*, *xsize=None*, *ysize=None*, *resampling=Resampling.cubic*, *masked=None*, *affine=None*)

Get window from raster.

Parameters

- **window** – requested window
- **bands** – list of indices of requested bads, default None which returns all bands
- **xsize** – tile x size default None, for full resolution pass None
- **ysize** – tile y size default None, for full resolution pass None
- **resampling** – which Resampling to use on reading, default Resampling.cubic
- **masked** – if True uses the maks, if False doesn't use the mask, if None looks to see if there is a mask, if mask exists using it, the default None

Returns
GeoRaster2 of tile

property height

Raster height.

property image

Raster bitmap in numpy array.

image_corner(*corner*)

Return image corner in pixels, as shapely.Point.

intersect(*other*)

Pixels outside either raster are set nodata

mask(*vector*, *mask_shape_nodata=False*)

Set pixels outside vector as nodata.

Parameters

- **vector** – GeoVector, GeoFeature, FeatureCollection
- **mask_shape_nodata** – if True - pixels inside shape are set nodata, if False - outside shape is nodata

Returns

GeoRaster2

mask_by_value(*nodata*)

Return raster with a mask calculated based on provided value. Only pixels with value=nodata will be masked.

Parameters

nodata – value of the pixels that should be masked

Returns

GeoRaster2

not_loaded()

Return True if image is not loaded.

property num_bands

Raster number of bands.

classmethod open(*filename*, *band_names=None*, *lazy_load=True*, *mutable=False*, ***kwargs*)

Read a georaster from a file.

Parameters

- **filename** – url
- **band_names** – list of strings, or string. if None - will try to read from image, otherwise - these will be ['0', ..]
- **lazy_load** – if True - do not load anything

Returns

GeoRaster2

origin()

Return footprint origin in world coordinates, as GeoVector.

property overviews_factors

returns the overviews factors

pixel_crop(*bounds*, *xsize=None*, *ysize=None*, *window=None*, *masked=None*, *bands=None*, *resampling=Resampling.cubic*)

Crop raster outside vector (convex hull).

Parameters

- **bounds** – bounds of requester portion of the image in image pixels

- **xsize** – output raster width, None for full resolution
- **ysize** – output raster height, None for full resolution
- **windows** – the bounds representation window on image in image pixels, Optional
- **bands** – list of indices of requested bands, default None which returns all bands
- **resampling** – reprojection resampling method, default *cubic*

Returns
GeoRaster

project(*dst_crs*, *resampling*)

Return reprojected raster.

rectify()

Rotate raster northwards.

reduce(*op*)

Reduce the raster to a score, using 'op' operation.

nodata pixels are ignored. op is currently limited to numpy.ma, e.g. 'mean', 'std' etc :returns list of per-band values

reproject(*dst_crs=None*, *resolution=None*, *dimensions=None*, *src_bounds=None*, *dst_bounds=None*, *rpcs=None*, *target_aligned_pixels=False*, *resampling=Resampling.cubic*, *creation_options=None*, ***kwargs*)

Return re-projected raster to new raster.

Parameters

- **dst_crs** (*rasterio.crs.CRS, optional*) – Target coordinate reference system.
- **resolution** (*tuple (x resolution, y resolution) or float, optional*) – Target resolution, in units of target coordinate reference system.
- **dimensions** (*tuple (width, height), optional*) – Output size in pixels and lines.
- **src_bounds** (*tuple (xmin, ymin, xmax, ymax), optional*) – Georeferenced extent of output (in source georeferenced units).
- **dst_bounds** (*tuple (xmin, ymin, xmax, ymax), optional*) – Georeferenced extent of output (in destination georeferenced units).
- **rpcs** (*RPC or dict, optional*) – Rational polynomial coefficients for the source.
- **target_aligned_pixels** (*bool, optional*) – Align the output bounds based on the resolution. Default is *False*.
- **resampling** (*rasterio.enums.Resampling*) – Reprojection resampling method. Default is *cubic*.
- **creation_options** (*dict, optional*) – Custom creation options.
- **kwargs** (*optional*) – Additional arguments passed to transformation function.

Returns
out

Return type
GeoRaster2

res_xy()

Returns X and Y resolution.

resize(*ratio=None*, *ratio_x=None*, *ratio_y=None*, *dest_width=None*, *dest_height=None*, *dest_resolution=None*, *resampling=Resampling.cubic*)

Provide either ratio, or ratio_x and ratio_y, or dest_width and/or dest_height.

Returns
GeoRaster2

resolution()

Return resolution. if different in different axis - return geometric mean.

property rpcs

Raster rpcs.

save(*filename*, *tags=None*, ***kwargs*)

Save GeoRaster to a file.

Parameters

- **filename** – url
- **tags** – tags to add to default namespace

optional parameters:

- GDAL_TIFF_INTERNAL_MASK: specifies whether mask is within image file, or additional .msk
- overviews: if True, will save with previews. default: True
- factors: list of factors for the overview, default: calculated based on raster width and height
- resampling: to build overviews. default: cubic
- tiled: if True raster will be saved tiled, default: False
- compress: any supported rasterio.enums.Compression value, default to LZW
- blockxsize: int, tile x size, default:256
- blockysize: int, tile y size, default:256
- creation_options: dict, key value of additional creation options
- nodata: if passed, will save with nodata value (e.g. useful for qgis)

save_cloud_optimized(*dest_url*, *resampling=Resampling.gauss*, *blocksize=256*, *overview_blocksize=256*, *creation_options=None*)

Save as Cloud Optimized GeoTiff object to a new file.

Parameters

- **dest_url** – path to the new raster
- **resampling** – which Resampling to use on reading, default Resampling.gauss
- **blocksize** – the size of the blocks default 256
- **overview_blocksize** – the block size of the overviews, default 256
- **creation_options** – dict, options that can override the source raster profile, notice that you can't override tiled=True, and the blocksize the list of creation_options can be found here https://www.gdal.org/frmt_gtiff.html

Returns
new GeoRaster of the tiled object

`property shape`

Raster shape.

`property source_file`

When using open, returns the filename used

`classmethod tags`(*filename*, *namespace=None*)

Extract tags from file.

`to_bytes`(*transparent=True*, *thumbnail_size=None*, *resampling=None*, *in_range='dtype'*, *out_range='dtype'*, *format='png'*)

Convert to selected format (discarding geo).

Optionally also resizes. Note: for color images returns interlaced. :param transparent: if True - sets alpha channel for nodata pixels :param thumbnail_size: if not None - resize to thumbnail size, e.g. 512 :param in_range: input intensity range :param out_range: output intensity range :param format : str, image format, default "png" :param resampling: one of Resampling enums

:return bytes

`to_pillow_image`(*return_mask=False*)

Return Pillow. Image, and optionally also mask.

`to_png`(*transparent=True*, *thumbnail_size=None*, *resampling=None*, *in_range='dtype'*, *out_range='dtype'*)

Convert to png format (discarding geo).

Optionally also resizes. Note: for color images returns interlaced. :param transparent: if True - sets alpha channel for nodata pixels :param thumbnail_size: if not None - resize to thumbnail size, e.g. 512 :param in_range: input intensity range :param out_range: output intensity range :param resampling: one of Resampling enums

:return bytes

`to_raster`(*vector*)

Return the vector in pixel coordinates, as shapely.Geometry.

`to_tiles`()

Yield slippy-map tiles.

`to_world`(*shape*, *dst_crs=None*)

Return the shape (provided in pixel coordinates) in world coordinates, as GeoVector.

`property transform`

Raster affine.

`vectorize`(*condition=None*)

Return GeoVector of raster, excluding nodata pixels, subject to 'condition'.

Parameters
condition – e.g. 42 < value < 142.

e.g. if no nodata pixels, and without condition - this == footprint().

`property width`

Raster width.

exception telluric.georaster.**GeoRaster2Error**

Base class for exceptions in the GeoRaster class.

exception telluric.georaster.**GeoRaster2IOError**

Base class for exceptions in GeoRaster read/write.

exception telluric.georaster.**GeoRaster2NotImplementedError**

Base class for NotImplementedError in the GeoRaster class.

exception telluric.georaster.**GeoRaster2Warning**

Base class for warnings in the GeoRaster class.

class telluric.georaster.**MergeStrategy**(*value*)

An enumeration.

class telluric.georaster.**MutableGeoRaster**(*image=None*, *affine=None*, *crs=None*, *filename=None*, *band_names=None*, *nodata=None*, *shape=None*, *footprint=None*, *rpcs=None*, *temporary=False*)

There are cases where you want to change the state of a *GeoRaster*, for these case conisder using *MutableGeoRaster*

This class allows you to change the following attributes:

- image - the entire image or the pixel in it
- band_names - the band_names count and the shape of the image must be consistent
- affine
- crs - we don't validate consistentency between affine and crs
- nodata_value

When mutable raster make sense:

- When you need to alter the the image and copying the image doesn't make sense
- When changing the affine or crs make sense without reprojecting

property **affine**

Raster affine.

property **band_names**

Raster affine.

property **crs**

Raster crs.

property **image**

Raster bitmap in numpy array.

class telluric.georaster.**PixelStrategy**(*value*)

An enumeration.

class telluric.georaster.**RasterChunk**(*raster*, *offsets*)

property **offsets**

Alias for field number 1

property **raster**

Alias for field number 0

telluric.georaster.**join**(*rasters*)

This method takes a list of rasters and returns a raster that is constructed of all of them

telluric.georaster.**merge_all**(*rasters*, *roi=None*, *dest_resolution=None*, *merge_strategy=MergeStrategy.UNION*, *shape=None*, *ul_corner=None*, *crs=None*, *pixel_strategy=PixelStrategy.FIRST*, *resampling=Resampling.nearest*, *crop=True*)

Merge a list of rasters, cropping (optional) by a region of interest. There are cases that the roi is not precise enough for this cases one can use, the upper left corner the shape and crs to precisely define the roi. When roi is provided the ul_corner, shape and crs are ignored.

NB: Reading rotated rasters with GDAL (and rasterio) gives unpredictable result and in order to overcome this you must use the warping algorithm to apply the rotation (it might be acomplished by using gdalwarp utility). Hence we should have the possibility to disable cropping, otherwise calling merge_all on rotated rasters may cause fails.

telluric.georaster.**merge_two**(*one:* GeoRaster2, *other:* GeoRaster2, *merge_strategy:* MergeStrategy = *MergeStrategy.UNION*, *silent:* *bool* = *False*, *pixel_strategy:* PixelStrategy = *PixelStrategy.FIRST*) → *GeoRaster2*

Merge two rasters into one.

Parameters

- **one** (GeoRaster2) – Left raster to merge.
- **other** (GeoRaster2) – Right raster to merge.
- **merge_strategy** (MergeStrategy, *optional*) – Merge strategy, from *telluric.georaster.MergeStrategy* (default to "union").
- **silent** (*bool*, *optional*) – Whether to raise errors or return some result, default to False (raise errors).
- **pixel_strategy** (PixelStrategy, *optional*) – Pixel strategy, from *telluric.georaster.PixelStrategy* (default to "top").

Return type
GeoRaster2

1.2.6 telluric.plotting module

Code for interactive vector plots.

telluric.plotting.**layer_from_element**(*element*, *style_function=None*)

Return Leaflet layer from shape.

Parameters
element (telluric.vectors.GeoVector, telluric.features.GeoFeature, telluric.collections.BaseCollection) – Data to plot.

telluric.plotting.**plot**(*feature*, *mp=None*, *style_function=None*, ***map_kwargs*)

Plots a GeoVector in an ipyleaflet map.

Parameters

- **feature** (telluric.vectors.GeoVector, telluric.features.GeoFeature, telluric.collections.BaseCollection) – Data to plot.
- **mp** (*ipyleaflet.Map*, *optional*) – Map in which to plot, default to None (creates a new one).

- **style_function** (*func*) – Function that returns an style dictionary for
- **map_kwargs** (*kwargs, optional*) – Extra parameters to send to ipyleaflet.Map.

telluric.plotting.**simple_plot**(*feature*, *, *mp=None*, ***map_kwargs*)

Plots a GeoVector in a simple Folium map.

For more complex and customizable plots using Jupyter widgets, use the plot function instead.

Parameters

feature (telluric.vectors.GeoVector, telluric.features.GeoFeature, telluric.collections.BaseCollection) – Data to plot.

telluric.plotting.**zoom_level_from_geometry**(*geometry*, *splits=4*)

Generate optimum zoom level for geometry.

Notes

The obvious solution would be

```
>>> mercantile.bounding_tile(*geometry.get_shape(WGS84_CRS).bounds).z
```

However, if the geometry is split between two or four tiles, the resulting zoom level might be too big.

1.2.7 telluric.util package

telluric.util.raster_utils.**build_overviews**(*source_file*, *factors=None*, *minsize=256*, *external=False*, *blocksize=256*, *interleave='pixel'*, *compress='lzw'*, *resampling=Resampling.gauss*, ***kwargs*)

Build overviews at one or more decimation factors for all bands of the dataset.

Parameters

- **source_file** (*str, file object or pathlib.Path object*) – Source file.
- **factors** (*list, optional*) – A list of integral overview levels to build.
- **minsize** (*int, optional*) – Maximum width or height of the smallest overview level. Only taken into account if explicit factors are not specified. Defaults to *256*.
- **external** (*bool, optional*) – Can be set to *True* to force external overviews in the GeoTIFF (.ovr) format. Default is False.
- **blocksize** (*int, optional*) – The block size (tile width and height) used for overviews. Should be a power-of-two value between 64 and 4096. Default value is *256*.
- **interleave** (*str, optional*) – Interleaving. Default value is *pixel*.
- **compress** (*str, optional*) – Set the compression to use. Default is *lzw*.
- **resampling** (*rasterio.enums.Resampling*) – Resampling method. Default is *gauss*.
- **kwargs** (*optional*) – Additional arguments passed to rasterio.Env.

Returns

out – Original file is altered or external .ovr can be created.

Return type

None

telluric.util.raster_utils.**build_vrt**(*source_file*, *destination_file*, ***kwargs*)

Make a VRT XML document and write it in file.

Parameters

- **source_file** (*str, file object or pathlib.Path object*) – Source file.
- **destination_file** (*str*) – Destination file.
- **kwargs** (*optional*) – Additional arguments passed to rasterio.vrt._boundless_vrt_doc

Returns

out – The path to the destination file.

Return type

str

telluric.util.raster_utils.**calc_transform**(*src*, *dst_crs=None*, *resolution=None*, *dimensions=None*, *rpcs=None*, *src_bounds=None*, *dst_bounds=None*, *target_aligned_pixels=False*, ***kwargs*)

Output dimensions and transform for a reprojection.

Parameters

- **src** (*rasterio.io.DatasetReader*) – Data source.
- **dst_crs** (*rasterio.crs.CRS, optional*) – Target coordinate reference system.
- **resolution** (*tuple (x resolution, y resolution) or float, optional*) – Target resolution, in units of target coordinate reference system.
- **dimensions** (*tuple (width, height), optional*) – Output file size in pixels and lines.
- **rpcs** (*RPC or dict, optional*) – Rational polynomial coefficients for the source.
- **src_bounds** (*tuple (xmin, ymin, xmax, ymax), optional*) – Georeferenced extent of output file from source bounds (in source georeferenced units).
- **dst_bounds** (*tuple (xmin, ymin, xmax, ymax), optional*) – Georeferenced extent of output file from destination bounds (in destination georeferenced units).
- **target_aligned_pixels** (*bool, optional*) – Align the output bounds based on the resolution. Default is *False*.
- **kwargs** (*optional*) – Additional arguments passed to transformation function.

Returns

- **dst_crs** (*rasterio.crs.CRS*) – Output crs
- **transform** (*Affine*) – Output affine transformation matrix
- **width, height** (*int*) – Output dimensions

telluric.util.raster_utils.**convert_to_cog**(*source_file*, *destination_file*, *resampling=Resampling.gauss*, *blocksize=256*, *overview_blocksize=256*, *creation_options=None*)

Convert source file to a Cloud Optimized GeoTiff new file.

Parameters

- **source_file** – path to the original raster
- **destination_file** – path to the new raster

- **resampling** – which Resampling to use on reading, default Resampling.gauss
- **blocksize** – the size of the blocks default 256
- **overview_blocksize** – the block size of the overviews, default 256
- **creation_options** – <dictioanry>, options that can override the source raster profile, notice that you can't override tiled=True, and the blocksize

telluric.util.raster_utils.**warp**(*source_file*, *destination_file*, *dst_crs=None*, *resolution=None*, *dimensions=None*, *src_bounds=None*, *dst_bounds=None*, *src_nodata=None*, *dst_nodata=None*, *rpcs=None*, *target_aligned_pixels=False*, *check_invert_proj=True*, *creation_options=None*, *resampling=Resampling.cubic*, ***kwargs*)

Warp a raster dataset.

Parameters

- **source_file** (*str, file object or pathlib.Path object*) – Source file.
- **destination_file** (*str, file object or pathlib.Path object*) – Destination file.
- **dst_crs** (*rasterio.crs.CRS, optional*) – Target coordinate reference system.
- **resolution** (*tuple (x resolution, y resolution) or float, optional*) – Target resolution, in units of target coordinate reference system.
- **dimensions** (*tuple (width, height), optional*) – Output file size in pixels and lines.
- **src_bounds** (*tuple (xmin, ymin, xmax, ymax), optional*) – Georeferenced extent of output file from source bounds (in source georeferenced units).
- **dst_bounds** (*tuple (xmin, ymin, xmax, ymax), optional*) – Georeferenced extent of output file from destination bounds (in destination georeferenced units).
- **src_nodata** (*int, float, or nan, optional*) – Manually overridden source nodata.
- **dst_nodata** (*int, float, or nan, optional*) – Manually overridden destination nodata.
- **rpcs** (*RPC or dict, optional*) – Rational polynomial coefficients for the source.
- **target_aligned_pixels** (*bool, optional*) – Align the output bounds based on the resolution. Default is *False*.
- **check_invert_proj** (*bool, optional*) – Constrain output to valid coordinate region in dst_crs. Default is *True*.
- **creation_options** (*dict, optional*) – Custom creation options.
- **resampling** (*rasterio.enums.Resampling*) – Reprojection resampling method. Default is *cubic*.
- **kwargs** (*optional*) – Additional arguments passed to transformation function.

Returns

out – Output is written to destination.

Return type

None

1.3 Changelog

1.3.1 telluric 0.14.4 (2023-04-18)

- Remove a deprecation warning (#326)

1.3.2 telluric 0.14.3 (2023-01-11)

- Fix memory leak (#262)

1.3.3 telluric 0.14.2 (2022-11-17)

- Rasterio 1.3.x compatibility, add Python 3.10 support (#323)

1.3.4 telluric 0.14.1 (2022-10-06)

- Don't raise an exception while calling *telluric.georaster.GeoRaster2.rpcs()* (#322)

1.3.5 telluric 0.14.0 (2022-02-17)

- Add RPCs support (#317)

1.3.6 telluric 0.13.14 (2022-01-28)

- Remove invalid creation options (#314, #315)

1.3.7 telluric 0.13.13 (2021-12-14)

- Support saving *GeoRaster2* with empty `affine` and `crs` (#312)
- Minor fixes

1.3.8 telluric 0.13.12 (2021-11-16)

- Use non-zero precision for windows rounding (#311)

1.3.9 telluric 0.13.11 (2021-10-19)

Changes

- Add <UseMaskBand>true</UseMaskBand> element to intermediate VRTs (#309)

1.3.10 telluric 0.13.10 (2021-10-10)

Changes

- Add Python 3.9 support (#303)
- Make *telluric.georaster.GeoRaster2.copy_with()* and dependant methods conserve mutability by default (#305)
- Remove pyproj dependency (#307)

1.3.11 telluric 0.13.9 (2021-07-01)

Changes

- Switch to Rasterio 1.2.0 and higher (#302)

1.3.12 telluric 0.13.8 (2021-06-13)

Changes

- Fix *telluric.georaster.GeoRaster2.save()* in order to take into account `creation_options` argument (#301)

1.3.13 telluric 0.13.7 (2021-05-13)

Changes

- Add `try - except` clauses in the call to `telluric.georaster._prepare_other_raster()` inside `telluric.georaster._prepare_rasters()` to let the rest of the rasters merge even if an exception is raised because some rasters footprints intersect but they can't be cropped (#299)

1.3.14 telluric 0.13.6 (2021-04-23)

Changes

- Check before calling `reproject` if the cropped raster has 0 width or height in `telluric.georaster._prepare_other_raster()` to avoid an exception for some corner cases when calling *telluric.georaster.merge_all()* (#296)

1.3.15 telluric 0.13.5 (2021-03-16)

Changes

- Fix memory leak in temporal rasters creation and deletion of `telluric.georaster.GeoRaster2._as_in_memory_geotiff()` (#294)

1.3.16 telluric 0.13.4 (2021-02-23)

Changes

- Set `_dtype` attribute in image setter for *MutableGeoRaster* (#289)
- Set `crs` as empty `rasterio.crs.CRS()` instance instead of `None` when image file has no CRS (#292)
- Make *telluric.georaster.GeoRaster2.resize()* faster (#293)

1.3.17 telluric 0.13.3 (2021-02-15)

Changes

- Add `crop` parameter to *telluric.georaster.merge_all()* function (#288)

1.3.18 telluric 0.13.2 (2020-11-27)

Changes

- Fix more imports when visualization dependencies are not installed (#283)

1.3.19 telluric 0.13.1 (2020-11-26)

Changes

- Fix imports when visualization dependencies are not installed (#281)
- Remove several deprecation warnings (#281)

1.3.20 telluric 0.13.0 (2020-11-25)

Changes

- Make visualization dependencies optional (#260)

1.3.21 telluric 0.12.1 (2020-08-10)

Bug fixes

- Check if the raster's footprint intersects the tile's footprint in *telluric.georaster.GeoRaster2.get_tile()* (#273)

1.3.22 telluric 0.12.0 (2020-08-02)

New features

- Preserve nodata value while saving rasters (#271)
- FileCollection created out of file-like object can be iterated (#272)

1.3.23 telluric 0.11.1 (2020-06-27)

Bug fixes

- Fix `telluric.collections.FileCollection.sort()` (#259)
- Fix potential bug in `ThreadContext` when it is uninitialized (#259)
- Disable transformation if source CRS equals to destination (#270)

1.3.24 telluric 0.11.0 (2019-12-02)

New features

- Now *MutableGeoRaster* inherits `nodata_value`

1.3.25 telluric 0.10.8 (2019-08-30)

Bug fixes

- Now reprojection retains nodata values

1.3.26 telluric 0.10.7 (2019-06-06)

New features

- Adding support of resources accesed through HTTP and HTTPS to VRT (#248)

Big fixes

- Remove unnecessary call of `fiona.Env` (#247)

1.3.27 telluric 0.10.6 (2019-05-02)

New features

- Creating COG with internal mask (#244)
- Removed pinning for pyproj (#245)

1.3.28 telluric 0.10.5 (2019-04-08)

Bug fixes

- Workaround to overcome impossible transformations (#241)

1.3.29 telluric 0.10.4 (2019-03-17)

Bug fixes

- Prevent image loading while copying (#235)

New features

- Refactored raster join implementation (#230)
- Changed default value of "nodata" in *GeoRaster2* constructor, now it is `None` (#231)
- Accelerate tests (#232)
- Added new method *telluric.georaster.GeoRaster2.mask_by_value()* (#233)
- Added new method *telluric.vectors.GeoVector.from_record()* (#238)
- Rasterio 1.0.21 compatibility (#239)
- Adding support to lazy resize that can use overviews if exist (#240)

1.3.30 telluric 0.10.3 (2019-01-10)

Bug fixes

- Fix *FeatureCollection* plotting (#229)

1.3.31 telluric 0.10.2 (2019-01-10)

New features

- SpatioTemporal Asset Catalog (STAC) compatibility (#223)
- Support custom schema in *telluric.collections.BaseCollection.save()* (#224)

Bug fixes

- Preserve the original schema while using *telluric.collections.BaseCollection.apply()* and *telluric.collections.BaseCollection.groupby()* (#225)
- Better handling of an empty collections (#226)
- Remove the reference to the raster object in the asset entry (#227)
- Retrieve mask in a safer way to avoid shrunk masks (#228)

1.3.32 telluric 0.10.1 (2018-12-27)

Bug fixes

- Fix masking by *GeoFeature* (#216)
- Fix issue in `GeoRaster.from_asset()` (#217, #220)
- `telluric.features.GeoFeature.envelope()` returns instance of *GeoVector* (#218)
- Use local tile server for visualization of `GeoFeatureWithRaster` (#221)
- *telluric.georaster.GeoRaster2.mask()* uses crop internally to reduce memory footprint (#219)
- `telluric.georaster.GeoRaster2.limit_to_bands()` is lazy (#222)

1.3.33 telluric 0.10.0 (2018-12-21)

New features

- Fiona 1.8.4 and Rasterio 1.0.13 compatibility (#207, #208)
- Support multiple rasters in a single `GeoFeatureWithRaster` (#209)
- Added new method *telluric.vectors.GeoVector.get_bounding_box()* (#213)

Bug fixes

- Remove hardcoded tile server port (#205)
- The internal state of the raster is not changed while saving (#210)
- Fix *telluric.georaster.GeoRaster2.save()* (#211)
- Fix bug in reproject (#212)
- Better handling of *telluric.features.GeoFeature.from_record()* (#214)

1.3.34 telluric 0.9.1 (2018-12-14)

New features

- LZW compression is used by default for creating COG rasters (#200)
- Added way to change port for local tile server (#202)

Bug fixes

- Fix iterating over *FileCollection* (#203)
- Fix fiona's GDAL environment issue (#204)

1.3.35 telluric 0.9.0 (2018-12-12)

New features

- Added new method *telluric.collections.FeatureCollection.from_georasters()* to create collections of rasters (#184)
- Visualization feature collection with rasters in Jupyter Notebook (#186)
- Added new method *telluric.collections.BaseCollection.apply()* (#188)
- Added new method *telluric.georaster.GeoRaster2.from_wms()* for creating rasters out of web services (#190, #192)
- Generalizing the process of making VRT files (#191, #193)
- Rasterio 1.0.11 compatibility (#194)
- Added new method *telluric.georaster.GeoRaster2.from_rasters()* to create raster out of a list of rasters (#195)
- Added support of several domains in a single VRT file (#196)

Bug fixes

- Reproject features before polygonization (#182)
- Fix `matplotlib.cm` call (#187)
- Fix *telluric.georaster.GeoRaster2.save()* (#197)
- Pin minimal version of Folium (#198)
- Fix rasterio's GDAL environment issue (#201)

1.3.36 telluric 0.8.0 (2018-11-18)

New features

- Interactive representation of rasters in Jupyter Notebook (#178)
- Fiona 1.8.1 and Rasterio 1.0.10 compatibility (#179, #180)

1.3.37 telluric 0.7.1 (2018-11-12)

Bug fixes

- Removed `pyplot` import from the module level to overcome issues at headless environments (#177)

1.3.38 telluric 0.7.0 (2018-11-06)

New features

- Added new method *telluric.georaster.GeoRaster2.chunks()* for iterating over the chunks of the raster (#169)

Bug fixes

- Workaround to overcome fiona's GDAL environment issue (#175)

1.3.39 telluric 0.6.0 (2018-11-05)

New features

- Added `resampling` parameter to *telluric.georaster.merge_all()* function (#166)
- New *telluric.vectors.GeoVector.tiles()* method for iterating over the tiles intersecting the bounding box of the vector (#167)
- Fiona 1.8.0 compatibility (#171)

Bug fixes

- Workaround to overcome rasterio's GDAL environment issue (#174)

1.3.40 telluric 0.5.0 (2018-10-26)

New features

- A new class *MutableGeoRaster* was added (#165)

1.3.41 telluric 0.4.1 (2018-10-23)

Bug fixes

- The right way to calculate `dest_resolution` in *telluric.georaster.merge_all()* if one is not provided (#163)
- Read mask only if it exists (#164)

1.3.42 telluric 0.4.0 (2018-10-19)

New features

- Rasterio 1.0.3 and higher compatibility (#152)
- Non-georeferenced images may be opened by providing `affine` and `crs` parameters to *telluric.georaster.GeoRaster2.open()* (#153)
- A new argument `crs` was added to *telluric.collections.FileCollection.open()* for opening vector files that dont't contain information about CRS (#156)
- A new *telluric.util.raster_utils.build_overviews()* utility was added (#158)

Bug fixes

- Treat 0 as legitimate value in *telluric.georaster.GeoRaster2.colorize()* (#160)
- Fix rasterization of an empty collection with callable `fill_value` (#161)

1.3.43 telluric 0.3.0 (2018-09-20)

New features

- New class `GeoFeatureWithRaster` that extends *GeoFeature*.

1.3.44 telluric 0.2.1 (2018-09-12)

Bug fixes

- Retrieve mask in a safer way in *telluric.georaster.GeoRaster2.save()* (#136)
- Fix affine calculation in *telluric.georaster.GeoRaster2.get_tile()* (#137)
- Convert dimensions to ints (#140)
- Masking areas outside the window in *telluric.georaster.GeoRaster2.get_window()* (#141)
- *telluric.georaster.merge_all()* does not crash for resolution in ROI units (#143, #146)
- Limit rasterio version to <1.0.3
- Add LICENSE into the MANIFEST (#147)

1.3.45 telluric 0.2.0 (2018-08-22)

New features

- Slicing a *FeatureCollection* now returns a `FeatureCollection` (#29, #32)
- Rasterization methods can now accept multiple fill values to produce nonbinary images (#34)
- `telluric.collections.FileCollection.save()` now saves types better (#20, #36)
- Merging functions and `telluric.georaster.GeoRaster2.empty_from_roi()` now support more ways to define the raster extent (#39, #57)

- Added utilities to convert to Cloud Optimized GeoTIFF (COG) and reproject files on disk (#45, #87)
- Raster data can be converted from/to different floating point formats thanks to enhancements in *telluric.georaster.GeoRaster2.astype()* (#33, #66)
- Added new method *telluric.georaster.GeoRaster2.colorize()* to colorize a band of a raster for visualization purposes (#81)
- Collections now have experimental "groupby/dissolve" functionality inspired by pandas and GeoPandas (#77, #98)
- Add a *telluric.georaster.PixelStrategy* enum with a new mode that allows the user to produce the "metadata" of a merge process (#68, #91)
- `telluric.vectors.GeoVector.rasterize()` can now accept a custom output CRS (#125)
- A new argument was added to the *GeoVector* constructor for disabling arguments validity checking (#126)
- Unnecessary CRS equality checking in *telluric.vectors.GeoVector.get_shape()* was removed for performance reasons (#127)

Deprecations and removals

- Rasterization methods no longer support specifying a "nodata" value, and an appropriate nodata value will be generated depending on the fill value(s) (#28, #34)
- Properties in the sense of the GeoJSON standard are now called "properties" instead of "attributes" for consistency (#84)
- Non georeferenced raster data is no longer supported (although we are considering re adding it under some restrictions) (#64, #74)
- It is not required for collections to be reprojected to output CRS for rasterization with *fill_value* (#125)

Bug fixes

- *telluric.vectors.GeoVector.from_record()* now treats `None` values properly (#37, #38)
- *GeoRaster2* methods and functions work with non isotropic resolution (#39)
- Cropping now behaves correctly with rasterio 1.0.0 (#44, #46)
- Crop size is now correctly computed for rasters in WGS84 (#61, #62)
- Fix rasterio 1.0.0 warnings regarding CRS comparison (#64, #74)
- *telluric.georaster.merge_all()* now is order independent and produces consistent results in all situations (#65, #62)
- *GeoRaster2* methods and functions work with rasters with positive y scale (#76, #78)
- *telluric.georaster.GeoRaster2.save()* with default arguments does not crash for small rasters anymore (#16, #53)
- `telluric.collections.FileCollection.save()` does not have side effects on heterogeneous collections anymore (#19, #24)
- Fix rasterization of points with default arguments (#9)

1.3.46 telluric 0.1.0 (2018-04-21)

Initial release

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

t

INDEX

Symbols

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P