

---

# **Telluric Documentation**

***Release v0.5.0***

**Juan Luis Cano, Slava Kerner, Lucio Torre**

**Oct 26, 2018**



---

## Contents:

---

<b>1</b>	<b>Installation</b>	<b>3</b>
1.1	User Guide . . . . .	3
1.2	API Reference . . . . .	9
1.3	Changelog . . . . .	24
<b>2</b>	<b>Indices and tables</b>	<b>27</b>
	<b>Python Module Index</b>	<b>29</b>



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.



# CHAPTER 1

---

## Installation

---

You can install telluric using pip:

```
pip install telluric
```

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

```
In [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:

```
In [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:

```
In [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"
```

---

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:

```
In [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:

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

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

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

```
Out[6]: 9422706289.175217
```

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

```
Out[7]: True
```

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

```
Out[8]: False
```

```
In [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"
```

## 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.

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

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

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

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

In [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'}))

In [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:

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

In [15]: !ls test_fc*

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

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

In [17]: !python -m json.tool < test_fc.json | head -n28
{
  "type": "FeatureCollection",
  "crs": {
    "type": "name",
    "properties": {
      "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:

```
In [18]: print(list(tl.FileCollection.open("test_fc.shp")))
[GeoFeature(Polygon, {'name': 'One feature', 'highlight': '{}', 'style': '{}'}), GeoFeature(Polygon,
```

### 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.

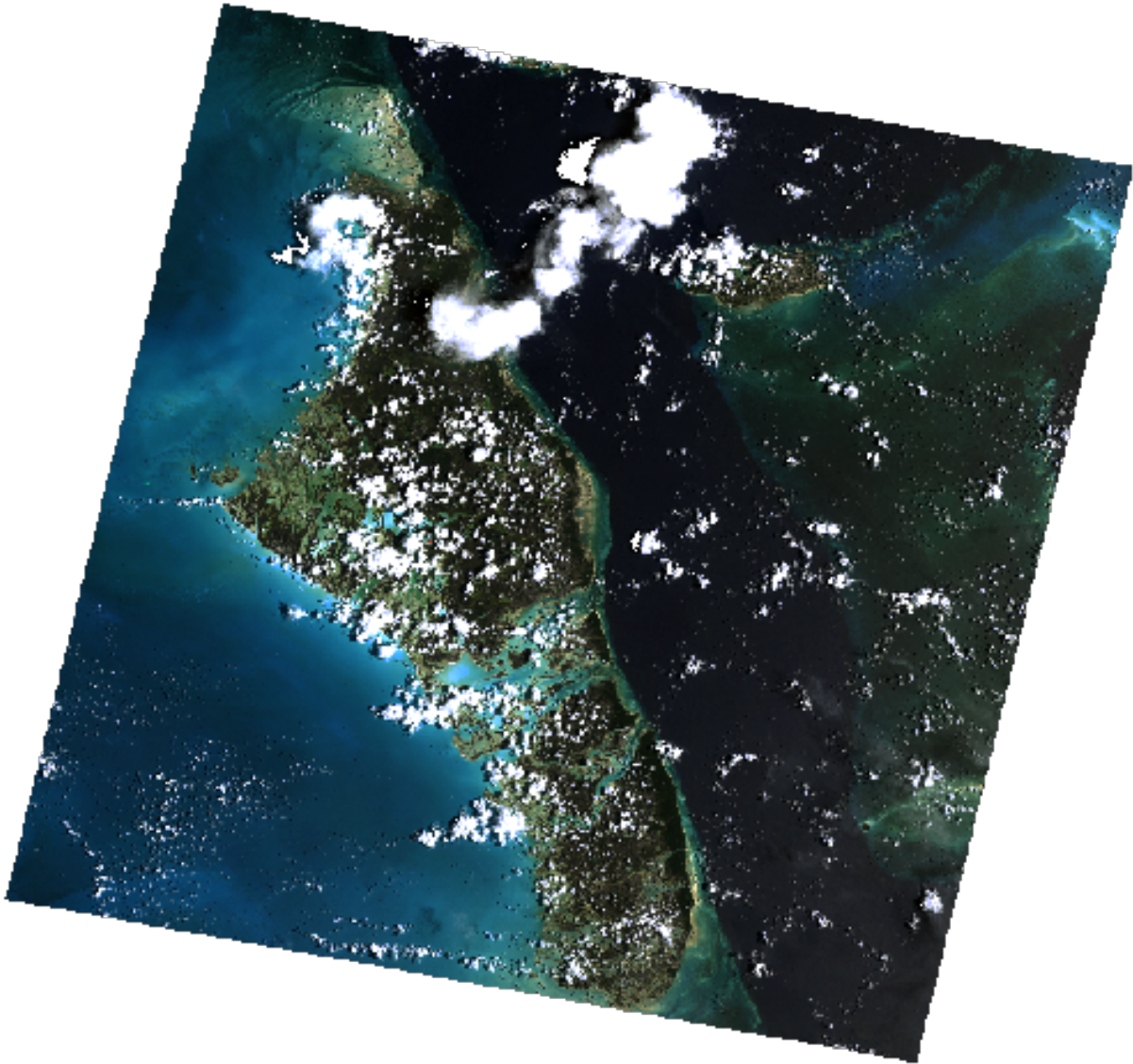
```
In [19]: # This will only save the URL in memory
rs = tl.GeoRaster2.open(
    "https://github.com/mapbox/rasterio/raw/master/tests/data/rgb_deflate.tif"
)

# These calls will fetch 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 26
[0, 1, 2]
```

`GeoRaster2` also displays itself automatically:

```
In [20]: rs
```

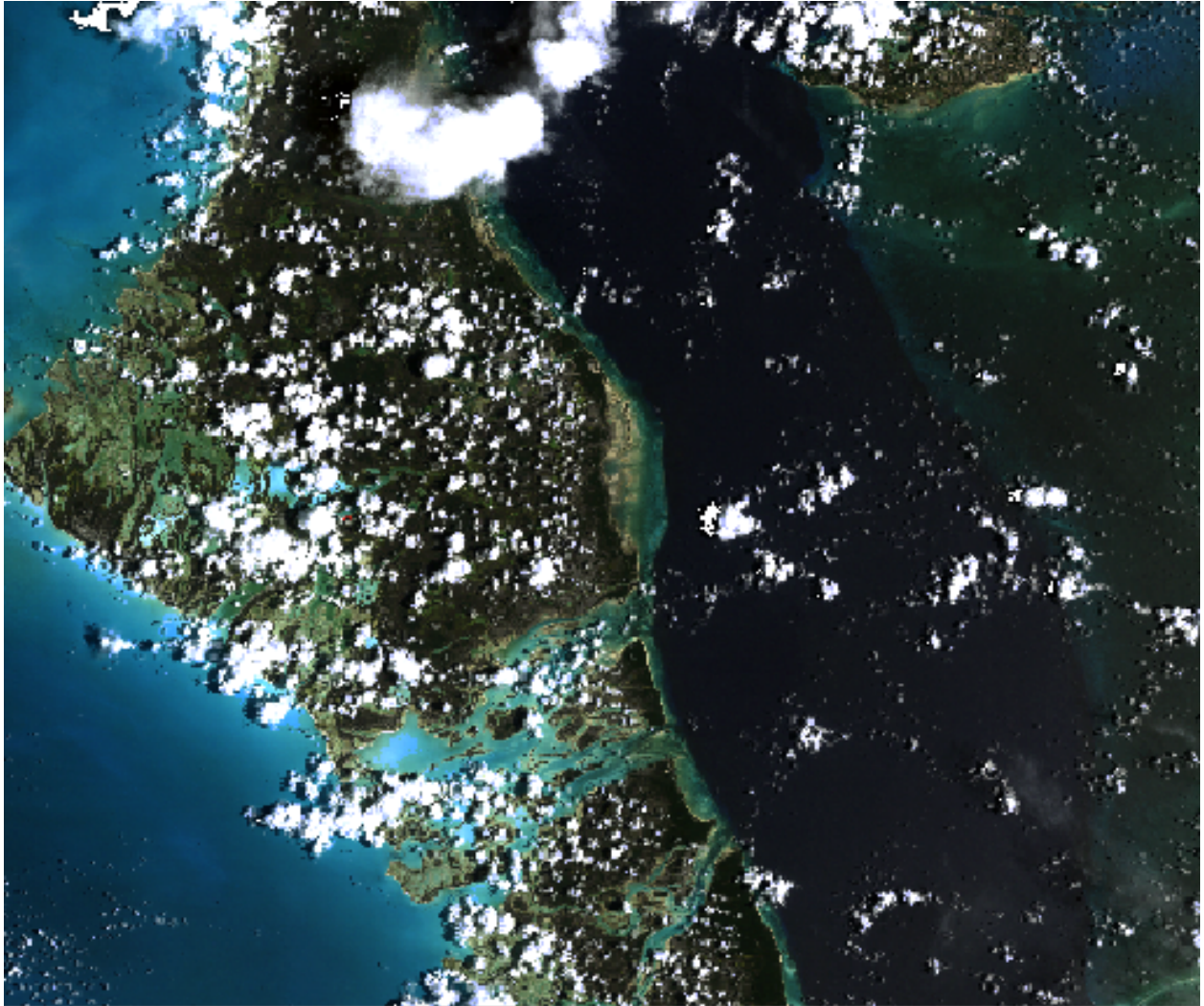


We can slice it like an array, or cropping some parts to discard others:

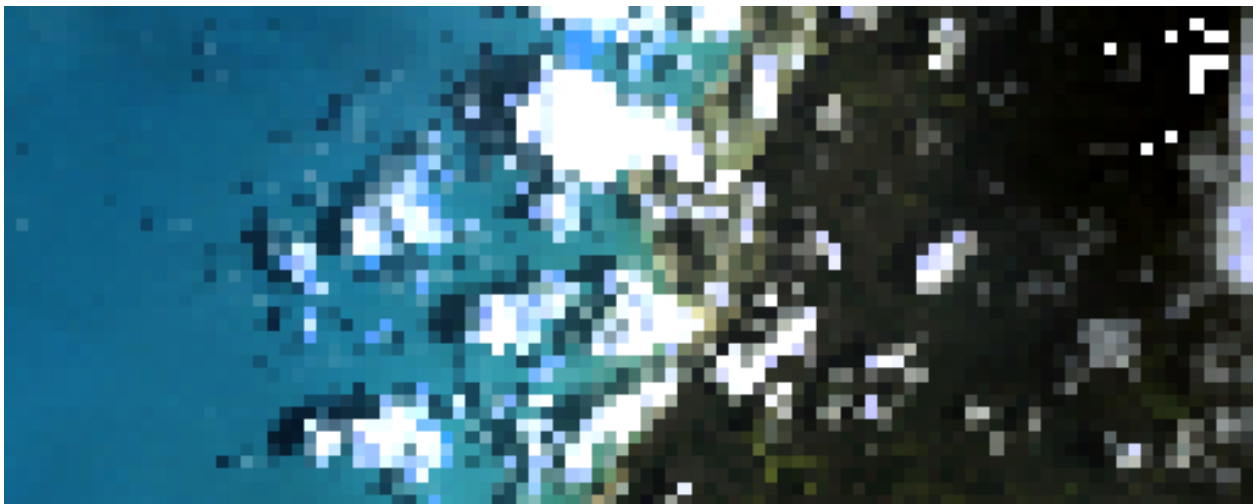
```
In [21]: rs.shape
```

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

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



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



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

```
In [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({'no_defs': True, 'init': 'epsg:4326'})
    Default CRS, set to WGS84_CRS.
```

```
telluric.constants.EQUAL_AREA_CRS = CRS({'proj': 'eck4'})
    Eckert IV CRS.
```

```
telluric.constants.WEB_MERCATOR_CRS = CRS({'no_defs': True, 'init': 'epsg:3857'})
    Web Mercator CRS.
```

```
telluric.constants.WGS84_CRS = CRS({'no_defs': True, 'init': 'epsg:4326'})
    WGS84 CRS.
```

### 1.2.2 telluric.vectors module

```
class telluric.vectors.GeoVector(shape, crs=CRS({'no_defs': True, 'init': '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({'no_defs': True, 'init': '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 (only CRS so far) if False, default to True

```
almost_equals(other, decimal=6)
    invariant to crs.
```

```
equals_exact(other, tolerance)
    invariant to crs.
```

```
classmethod from_bounds(xmin, ymin, xmax, ymax, crs=CRS({'no_defs': True, 'init':
                                                         'epsg:4326'}))
```

Creates GeoVector object from bounds.

#### Parameters

- **ymin, xmax, ymax** (*xmin,*) – 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\_xyz** (*x*, *y*, *z*)

Creates GeoVector from Mercator slippy map values.

**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.

**to\_geojson** (*filename*)

Save vector as geojson.

`telluric.vectors.generate_tile_coordinates` (*roi*, *num\_tiles*)

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*)

GeoFeature object.

`__init__(geovector, properties)`  
Initialize a GeoFeature object.

**Parameters**

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

**classmethod from\_raster** (*raster, properties*)  
Initialize a GeoFeature object with a GeoRaster

**Parameters**

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

**get\_shape** (*crs*)  
Gets the underlying Shapely shape in a specified CRS.

**class** telluric.features.**GeoFeatureWithRaster** (*raster, properties*)

`__init__(raster, properties)`

**Initialize a GeoFeature object with a raster**, When a GeoFeature has a raster the default behaviour is the same as GeoFeature where the geometry is the union of all rasters footprint.

we will override some methods to work differetnly like: 1. reproject (TBD) 2. rasterize (from feature collection, TBD) 3. to\_record

**Parameters**

- **raster[s]** (*array or single GeoRaster*) –
- **properties** (*dict*) – Properties.

telluric.features.**raster\_from\_assets** (*assets*)

create a raster from assets, assets is a dictionary of links like described in the stacs inteface <https://github.com/radianteearth/stac-spec/tree/master/json-spec/examples>

- currently we support a single raster asset, in the feature we could support more

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**

**dissolve** (*by=None, aggfunc=None*)

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*)

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.

**Returns**

**Return type** `_CollectionGroupBy`

**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({'no\_defs': True, 'init': '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*)

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 and return a value by which to sort.
- **desc** (*bool, optional*) – Descending sort, default to False (ascending).



```
class telluric.collections.FeatureCollection (results)
```

```
    __init__ (results)
```

Initialize FeatureCollection object.

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

```
    classmethod from_geovectors (geovectors)
```

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

```
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 function will not reproject

```
telluric.collections.dissolve (collection, aggfunc=None)
```

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

## 1.2.5 telluric.georaster module

```
class telluric.georaster.GeoRaster2 (image=None, affine=None, crs=None, filename=None,  
                                     band_names=None, nodata=0, shape=None, footprint=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=0,  
            shape=None, footprint=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
- **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.

**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/...

**band\_names**  
Raster affine.

**bounds** ()  
Return image rectangle in pixels, as shapely.Polygon.

**center** ()  
Return footprint center in world coordinates, as GeoVector.

**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](https://matplotlib.org/examples/color/colormaps_reference.html)

#### Parameters

- **colormap** (*str*) –
- **name from this list** [https](https://matplotlib.org/examples/color/colormaps_reference.html) (*Colormap*) –

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

**Returns****Return type** *GeoRaster2***copy\_with** (*mutable=False*, *\*\*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 -&gt; GeoVector}.

**crop** (*vector*, *resolution=None*, *masked=None*, *bands=None*)

crops raster outside vector (convex hull) :param vector: GeoVector :param resolution: output resolution, None for full resolution :return: GeoRaster

**crs**

Raster crs.

**deepcopy\_with** (*mutable=False*, *\*\*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\_tiles** (*tiles*)

Compose raster from tiles. return GeoRaster.

**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: 2>*)

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

**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: 2>*, *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

#### height

Raster height.

#### 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

#### 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.

**pixel\_crop** (*bounds, xsize=None, ysize=None, window=None, masked=None, bands=None*)

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

**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, target\_aligned\_pixels=False, resampling=<Resampling.cubic: 2>, 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).
- **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: 2>*)

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.

**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: 7>*)

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

**Returns** new VirtualGeoRaster of the tiled object

**shape**

Raster shape.

**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 sloppy-map tiles.

**to\_world** (*shape, dst\_crs=None*)

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

**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()`.

**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`

An enumeration.

**class** `telluric.georaster.MutableGeoRaster` (*image=None, affine=None, crs=None, filename=None, band\_names=None, no-data=0, shape=None, footprint=None, temporary=False*)

There are cases where you want to change the state of a *GeoRaster*, for these case consider 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 consistency between `affine` and `crs`

**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

**`affine`**

Raster affine.

**`band_names`**

Raster affine.

**`crs`**

Raster crs.

**`image`**

Raster bitmap in numpy array.

**`class telluric.georaster.PixelStrategy`**

An enumeration.

```
telluric.georaster.merge_all(rasters, roi=None, dest_resolution=None,
                             merge_strategy=<MergeStrategy.UNION: 2>, shape=None,
                             ul_corner=None, crs=None, pixel_strategy=<PixelStrategy.FIRST:
                             1>)
```

Merge a list of rasters, cropping 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

```
telluric.georaster.merge_two(one, other, merge_strategy=<MergeStrategy.UNION: 2>,
                             silent=False, pixel_strategy=<PixelStrategy.FIRST: 1>)
```

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”).

**Returns**

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: 7>*, *\*\*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.calc_transform(src, dst_crs=None, resolution=None, dimensions=None, src_bounds=None, dst_bounds=None, target_aligned_pixels=False)
```

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.
- **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*.

#### 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: 7>, 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, target_aligned_pixels=False, check_invert_proj=True, creation_options=None, resampling=<Resampling.cubic: 2>, **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.
- **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 0.5.0 (2018-10-26)

#### New features

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

### 1.3.2 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)

### 1.3.3 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 don'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)

### 1.3.4 telluric 0.3.0 (2018-09-20)

#### New features

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

### 1.3.5 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)

### 1.3.6 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)

## 1.3.7 telluric 0.1.0 (2018-04-21)

Initial release

## CHAPTER 2

---

### Indices and tables

---

- `genindex`
- `modindex`
- `search`





### t

- `telluric.collections`, [11](#)
- `telluric.constants`, [9](#)
- `telluric.features`, [10](#)
- `telluric.georaster`, [13](#)
- `telluric.plotting`, [20](#)
- `telluric.util.raster_utils`, [21](#)
- `telluric.vectors`, [9](#)



## Symbols

`__init__()` (telluric.collections.FeatureCollection method), 13  
`__init__()` (telluric.collections.FileCollection method), 13  
`__init__()` (telluric.features.GeoFeature method), 10  
`__init__()` (telluric.features.GeoFeatureWithRaster method), 11  
`__init__()` (telluric.georaster.GeoRaster2 method), 13  
`__init__()` (telluric.vectors.GeoVector method), 9

## A

`add_raster()` (telluric.georaster.GeoRaster2 method), 14  
`affine` (telluric.georaster.GeoRaster2 attribute), 14  
`affine` (telluric.georaster.MutableGeoRaster attribute), 20  
`almost_equals()` (telluric.vectors.GeoVector method), 9  
`apply_transform()` (telluric.georaster.GeoRaster2 method), 14  
`astype()` (telluric.georaster.GeoRaster2 method), 14  
`attributes()` (telluric.georaster.GeoRaster2 method), 14

## B

`band_names` (telluric.georaster.GeoRaster2 attribute), 14  
`band_names` (telluric.georaster.MutableGeoRaster attribute), 20  
`BaseCollection` (class in telluric.collections), 11  
`bounds()` (telluric.georaster.GeoRaster2 method), 14  
`build_overviews()` (in module telluric.util.raster\_utils), 21

## C

`calc_transform()` (in module telluric.util.raster\_utils), 22  
`center()` (telluric.georaster.GeoRaster2 method), 14  
`colorize()` (telluric.georaster.GeoRaster2 method), 14  
`convert_to_cog()` (in module telluric.util.raster\_utils), 22  
`copy_with()` (telluric.georaster.GeoRaster2 method), 15  
`corner()` (telluric.georaster.GeoRaster2 method), 15  
`corners()` (telluric.georaster.GeoRaster2 method), 15  
`crop()` (telluric.georaster.GeoRaster2 method), 15  
`crs` (telluric.georaster.GeoRaster2 attribute), 15  
`crs` (telluric.georaster.MutableGeoRaster attribute), 20

## D

`deepcopy_with()` (telluric.georaster.GeoRaster2 method), 15  
`DEFAULT_CRS` (in module telluric.constants), 9  
`dissolve()` (in module telluric.collections), 13  
`dissolve()` (telluric.collections.BaseCollection method), 11

## E

`EQUAL_AREA_CRS` (in module telluric.constants), 9  
`equals_exact()` (telluric.vectors.GeoVector method), 9

## F

`FeatureCollection` (class in telluric.collections), 13  
`FeatureCollectionIOError`, 13  
`FileCollection` (class in telluric.collections), 13  
`filter()` (telluric.collections.BaseCollection method), 12  
`from_bounds()` (telluric.vectors.GeoVector class method), 9  
`from_bytes()` (telluric.georaster.GeoRaster2 class method), 15  
`from_geojson()` (telluric.vectors.GeoVector class method), 10  
`from_geovectors()` (telluric.collections.FeatureCollection class method), 13  
`from_raster()` (telluric.features.GeoFeature class method), 11  
`from_tiles()` (telluric.georaster.GeoRaster2 class method), 15  
`from_xyz()` (telluric.vectors.GeoVector class method), 10

## G

`generate_tile_coordinates()` (in module telluric.vectors), 10  
`generate_tile_coordinates_from_pixels()` (in module telluric.vectors), 10  
`GeoFeature` (class in telluric.features), 10  
`GeoFeatureWithRaster` (class in telluric.features), 11  
`GeoRaster2` (class in telluric.georaster), 13

GeoRaster2Error, 19  
 GeoRaster2IOError, 19  
 GeoRaster2NotImplementedError, 19  
 GeoRaster2Warning, 19  
 GeoVector (class in telluric.vectors), 9  
 get() (telluric.georaster.GeoRaster2 method), 15  
 get\_dimension() (in module telluric.vectors), 10  
 get\_shape() (telluric.features.GeoFeature method), 11  
 get\_shape() (telluric.vectors.GeoVector method), 10  
 get\_tile() (telluric.georaster.GeoRaster2 method), 15  
 get\_values() (telluric.collections.BaseCollection method), 12  
 get\_window() (telluric.georaster.GeoRaster2 method), 16  
 groupby() (telluric.collections.BaseCollection method), 12

## H

height (telluric.georaster.GeoRaster2 attribute), 16

## I

image (telluric.georaster.GeoRaster2 attribute), 16  
 image (telluric.georaster.MutableGeoRaster attribute), 20  
 image\_corner() (telluric.georaster.GeoRaster2 method), 16  
 intersect() (telluric.georaster.GeoRaster2 method), 16  
 is\_empty (telluric.collections.BaseCollection attribute), 12

## L

layer\_from\_element() (in module telluric.plotting), 20

## M

map() (telluric.collections.BaseCollection method), 12  
 mask() (telluric.georaster.GeoRaster2 method), 16  
 merge\_all() (in module telluric.georaster), 20  
 merge\_two() (in module telluric.georaster), 20  
 MergeStrategy (class in telluric.georaster), 19  
 MutableGeoRaster (class in telluric.georaster), 19

## N

num\_bands (telluric.georaster.GeoRaster2 attribute), 16

## O

open() (telluric.collections.FileCollection class method), 13  
 open() (telluric.georaster.GeoRaster2 class method), 16  
 origin() (telluric.georaster.GeoRaster2 method), 17

## P

pixel\_crop() (telluric.georaster.GeoRaster2 method), 17  
 PixelStrategy (class in telluric.georaster), 20  
 plot() (in module telluric.plotting), 21  
 polygonize() (telluric.vectors.GeoVector method), 10

project() (telluric.georaster.GeoRaster2 method), 17

## R

raster\_from\_assets() (in module telluric.features), 11  
 rasterize() (telluric.collections.BaseCollection method), 12  
 rectify() (telluric.georaster.GeoRaster2 method), 17  
 reduce() (telluric.georaster.GeoRaster2 method), 17  
 reproject() (telluric.georaster.GeoRaster2 method), 17  
 res\_xy() (telluric.georaster.GeoRaster2 method), 18  
 resize() (telluric.georaster.GeoRaster2 method), 18  
 resolution() (telluric.georaster.GeoRaster2 method), 18

## S

save() (telluric.collections.BaseCollection method), 12  
 save() (telluric.georaster.GeoRaster2 method), 18  
 save\_cloud\_optimized() (telluric.georaster.GeoRaster2 method), 18  
 serialize\_properties() (in module telluric.features), 11  
 shape (telluric.georaster.GeoRaster2 attribute), 18  
 simple\_plot() (in module telluric.plotting), 21  
 sort() (telluric.collections.BaseCollection method), 12

## T

tags() (telluric.georaster.GeoRaster2 class method), 18  
 telluric.collections (module), 11  
 telluric.constants (module), 9  
 telluric.features (module), 10  
 telluric.georaster (module), 13  
 telluric.plotting (module), 20  
 telluric.util.raster\_utils (module), 21  
 telluric.vectors (module), 9  
 to\_bytes() (telluric.georaster.GeoRaster2 method), 18  
 to\_gejson() (telluric.vectors.GeoVector method), 10  
 to\_pillow\_image() (telluric.georaster.GeoRaster2 method), 19  
 to\_png() (telluric.georaster.GeoRaster2 method), 19  
 to\_raster() (telluric.georaster.GeoRaster2 method), 19  
 to\_tiles() (telluric.georaster.GeoRaster2 method), 19  
 to\_world() (telluric.georaster.GeoRaster2 method), 19  
 transform (telluric.georaster.GeoRaster2 attribute), 19  
 transform\_properties() (in module telluric.features), 11

## V

vectorize() (telluric.georaster.GeoRaster2 method), 19

## W

warp() (in module telluric.util.raster\_utils), 23  
 WEB\_MERCATOR\_CRS (in module telluric.constants), 9  
 WGS84\_CRS (in module telluric.constants), 9  
 width (telluric.georaster.GeoRaster2 attribute), 19

## Z

`zoom_level_from_geometry()` (in module `telluric.plotting`), [21](#)