
geojson Documentation

Release latest

Aug 10, 2020

1	Installation	3
2	GeoJSON Objects	5
2.1	Point	5
2.2	MultiPoint	5
2.3	LineString	6
2.4	MultiLineString	6
2.5	Polygon	6
2.6	MultiPolygon	6
2.7	GeometryCollection	7
2.8	Feature	7
2.9	FeatureCollection	8
3	GeoJSON encoding/decoding	9
3.1	Custom classes	9
3.2	Default and custom precision	10
4	Helpful utilities	11
4.1	coords	11
4.2	map_coords	11
4.3	map_tuples	12
4.4	map_geometries	12
4.5	validation	12
4.6	generate_random	12
5	Development	15
6	Credits	17

This Python library contains:

- Functions for encoding and decoding [GeoJSON](#) formatted data
- Classes for all GeoJSON Objects
- An implementation of the Python `__geo_interface__` Specification

Table of Contents

- *Installation*
- *GeoJSON Objects*
 - *Point*
 - *MultiPoint*
 - *LineString*
 - *MultiLineString*
 - *Polygon*
 - *MultiPolygon*
 - *GeometryCollection*
 - *Feature*
 - *FeatureCollection*
- *GeoJSON encoding/decoding*
 - *Custom classes*
 - *Default and custom precision*
- *Helpful utilities*
 - *coords*
 - *map_coords*
 - *map_tuples*
 - *map_geometries*
 - *validation*
 - *generate_random*
- *Development*
- *Credits*

CHAPTER 1

Installation

gejson is compatible with Python 3.6, 3.7 and 3.8. The recommended way to install is via [pip](#):

```
pip install gejson
```


This library implements all the [GeoJSON Objects](#) described in [The GeoJSON Format Specification](#).

All object keys can also be used as attributes.

The objects contained in `GeometryCollection` and `FeatureCollection` can be indexed directly.

2.1 Point

```
>>> from geojson import Point
>>> Point((-115.81, 37.24))
{"coordinates": [-115.8..., 37.2...], "type": "Point"}
```

Visualize the result of the example above [here](#). General information about `Point` can be found in [Section 3.1.2](#) and [Appendix A: Points within The GeoJSON Format Specification](#).

2.2 MultiPoint

```
>>> from geojson import MultiPoint
>>> MultiPoint([(-155.52, 19.61), (-156.22, 20.74), (-157.97, 21.46)])
{"coordinates": [[-155.5..., 19.6...], [-156.2..., 20.7...], [-157.9..., 21.4...]],
↪ "type": "MultiPoint"}
```

Visualize the result of the example above [here](#). General information about `MultiPoint` can be found in [Section 3.1.3](#) and [Appendix A: MultiPoints within The GeoJSON Format Specification](#).

2.3 LineString

```
>>> from geojson import LineString

>>> LineString([(8.919, 44.4074), (8.923, 44.4075)])
{"coordinates": [[8.91..., 44.407...], [8.92..., 44.407...]], "type": "LineString"}
```

Visualize the result of the example above [here](#). General information about LineString can be found in [Section 3.1.4](#) and [Appendix A: LineStrings within The GeoJSON Format Specification](#).

2.4 MultiLineString

```
>>> from geojson import MultiLineString

>>> MultiLineString([
...     [(3.75, 9.25), (-130.95, 1.52)],
...     [(23.15, -34.25), (-1.35, -4.65), (3.45, 77.95)]
... ])
{"coordinates": [[[3.7..., 9.2...], [-130.9..., 1.52...]], [[23.1..., -34.2...], [-1.
↪3..., -4.6...], [3.4..., 77.9...]]], "type": "MultiLineString"}
```

Visualize the result of the example above [here](#). General information about MultiLineString can be found in [Section 3.1.5](#) and [Appendix A: MultiLineStrings within The GeoJSON Format Specification](#).

2.5 Polygon

```
>>> from geojson import Polygon

>>> # no hole within polygon
>>> Polygon([(2.38, 57.322), (23.194, -20.28), (-120.43, 19.15), (2.38, 57.322)])
{"coordinates": [[[2.3..., 57.32...], [23.19..., -20.2...], [-120.4..., 19.1...]]],
↪ "type": "Polygon"}

>>> # hole within polygon
>>> Polygon([
...     [(2.38, 57.322), (23.194, -20.28), (-120.43, 19.15), (2.38, 57.322)],
...     [(-5.21, 23.51), (15.21, -10.81), (-20.51, 1.51), (-5.21, 23.51)]
... ])
{"coordinates": [[[2.3..., 57.32...], [23.19..., -20.2...], [-120.4..., 19.1...], [[-
↪5.2..., 23.5...], [15.2..., -10.8...], [-20.5..., 1.5...], [-5.2..., 23.5...]]],
↪ "type": "Polygon"}
```

Visualize the results of the example above [here](#). General information about Polygon can be found in [Section 3.1.6](#) and [Appendix A: Polygons within The GeoJSON Format Specification](#).

2.6 MultiPolygon

```
>>> from geojson import MultiPolygon
```

(continues on next page)

(continued from previous page)

```
>>> MultiPolygon([
...     ((3.78, 9.28), (-130.91, 1.52), (35.12, 72.234), (3.78, 9.28)),
...     ((23.18, -34.29), (-1.31, -4.61), (3.41, 77.91), (23.18, -34.29)),
... ])
{"coordinates": [[[3.7..., 9.2...], [-130.9..., 1.5...], [35.1..., 72.23...]]],
↳ [[23.1..., -34.2...], [-1.3..., -4.6...], [3.4..., 77.9...]]], "type":
↳ "MultiPolygon"}
```

Visualize the result of the example above [here](#). General information about MultiPolygon can be found in [Section 3.1.7](#) and [Appendix A: MultiPolygons within The GeoJSON Format Specification](#).

2.7 GeometryCollection

```
>>> from geojson import GeometryCollection, Point, LineString

>>> my_point = Point((23.532, -63.12))

>>> my_line = LineString((-152.62, 51.21), (5.21, 10.69))

>>> geo_collection = GeometryCollection([my_point, my_line])

>>> geo_collection
{"geometries": [{"coordinates": [23.53..., -63.1...], "type": "Point"}, {"coordinates
↳": [[-152.6..., 51.2...], [5.2..., 10.6...]], "type": "LineString"}], "type":
↳ "GeometryCollection"}

>>> geo_collection[1]
{"coordinates": [[-152.62, 51.21], [5.21, 10.69]], "type": "LineString"}

>>> geo_collection[0] == geo_collection.geometries[0]
True
```

Visualize the result of the example above [here](#). General information about GeometryCollection can be found in [Section 3.1.8](#) and [Appendix A: GeometryCollections within The GeoJSON Format Specification](#).

2.8 Feature

```
>>> from geojson import Feature, Point

>>> my_point = Point((-3.68, 40.41))

>>> Feature(geometry=my_point)
{"geometry": {"coordinates": [-3.68..., 40.4...], "type": "Point"}, "properties": {},
↳ "type": "Feature"}

>>> Feature(geometry=my_point, properties={"country": "Spain"})
{"geometry": {"coordinates": [-3.68..., 40.4...], "type": "Point"}, "properties": {
↳ "country": "Spain"}, "type": "Feature"}

>>> Feature(geometry=my_point, id=27)
{"geometry": {"coordinates": [-3.68..., 40.4...], "type": "Point"}, "id": 27,
↳ "properties": {}, "type": "Feature"}
```

Visualize the results of the examples above [here](#). General information about Feature can be found in [Section 3.2](#) within [The GeoJSON Format Specification](#).

2.9 FeatureCollection

```
>>> from geojson import Feature, Point, FeatureCollection
>>> my_feature = Feature(geometry=Point((1.6432, -19.123)))
>>> my_other_feature = Feature(geometry=Point((-80.234, -22.532)))
>>> feature_collection = FeatureCollection([my_feature, my_other_feature])
>>> feature_collection
{"features": [{"geometry": {"coordinates": [1.643..., -19.12...], "type": "Point"},
↪ "properties": {}, "type": "Feature"}, {"geometry": {"coordinates": [-80.23..., -22.
↪ 53...], "type": "Point"}, "properties": {}, "type": "Feature"}], "type":
↪ "FeatureCollection"}
>>> feature_collection.errors()
[]
>>> (feature_collection[0] == feature_collection['features'][0], feature_
↪ collection[1] == my_other_feature)
(True, True)
```

Visualize the result of the example above [here](#). General information about FeatureCollection can be found in [Section 3.3](#) within [The GeoJSON Format Specification](#).

GeoJSON encoding/decoding

All of the GeoJSON Objects implemented in this library can be encoded and decoded into raw GeoJSON with the `geojson.dump`, `geojson.dumps`, `geojson.load`, and `geojson.loads` functions. Note that each of these functions is a wrapper around the core `json` function with the same name, and will pass through any additional arguments. This allows you to control the JSON formatting or parsing behavior with the underlying core `json` functions.

```
>>> import geojson

>>> my_point = geojson.Point((43.24, -1.532))

>>> my_point
{"coordinates": [43.2..., -1.53...], "type": "Point"}

>>> dump = geojson.dumps(my_point, sort_keys=True)

>>> dump
'{"coordinates": [43.2..., -1.53...], "type": "Point"}'

>>> geojson.loads(dump)
{"coordinates": [43.2..., -1.53...], "type": "Point"}
```

3.1 Custom classes

This encoding/decoding functionality shown in the previous can be extended to custom classes using the interface described by the `__geo_interface__` Specification.

```
>>> import geojson

>>> class MyPoint():
...     def __init__(self, x, y):
...         self.x = x
...         self.y = y
```

(continues on next page)

(continued from previous page)

```

...
...     @property
...     def __geo_interface__(self):
...         return {'type': 'Point', 'coordinates': (self.x, self.y)}

>>> point_instance = MyPoint(52.235, -19.234)

>>> geojson.dumps(point_instance, sort_keys=True)
'{"coordinates": [52.23..., -19.23...], "type": "Point"}'

```

3.2 Default and custom precision

GeoJSON Object-based classes in this package have an additional *precision* attribute which rounds off coordinates to 6 decimal places (roughly 0.1 meters) by default and can be customized per object instance.

```

>>> from geojson import Point

>>> Point((-115.123412341234, 37.123412341234)) # rounded to 6 decimal places by_
↳ default
{"coordinates": [-115.123412, 37.123412], "type": "Point"}

>>> Point((-115.12341234, 37.12341234), precision=8) # rounded to 8 decimal places
{"coordinates": [-115.12341234, 37.12341234], "type": "Point"}

```

4.1 coords

`geojson.utils.coords` yields all coordinate tuples from a geometry or feature object.

```
>>> import geojson
>>> my_line = LineString([(-152.62, 51.21), (5.21, 10.69)])
>>> my_feature = geojson.Feature(geometry=my_line)
>>> list(geojson.utils.coords(my_feature))
[(-152.62..., 51.21...), (5.21..., 10.69...)]
```

4.2 map_coords

`geojson.utils.map_coords` maps a function over all coordinate values and returns a geometry of the same type. Useful for scaling a geometry.

```
>>> import geojson
>>> new_point = geojson.utils.map_coords(lambda x: x/2, geojson.Point((-115.81, 37.
↪24)))
>>> geojson.dumps(new_point, sort_keys=True)
'{"coordinates": [-57.905..., 18.62...], "type": "Point"}'
```

4.3 map_tuples

`geojson.utils.map_tuples` maps a function over all coordinates and returns a geometry of the same type. Useful for changing coordinate order or applying coordinate transforms.

```
>>> import geojson

>>> new_point = geojson.utils.map_tuples(lambda c: (c[1], c[0]), geojson.Point((-115.
↪81, 37.24)))

>>> geojson.dumps(new_point, sort_keys=True)
'{"coordinates": [37.24..., -115.81], "type": "Point"}'
```

4.4 map_geometries

`geojson.utils.map_geometries` maps a function over each geometry in the input.

```
>>> import geojson

>>> new_point = geojson.utils.map_geometries(lambda g: geojson.MultiPoint([g[
↪"coordinates"]]), geojson.GeometryCollection([geojson.Point((-115.81, 37.24))]))

>>> geojson.dumps(new_point, sort_keys=True)
'{"geometries": [{"coordinates": [[-115.81, 37.24]], "type": "MultiPoint"}], "type":
↪"GeometryCollection}"'
```

4.5 validation

`is_valid` property provides simple validation of GeoJSON objects.

```
>>> import geojson

>>> obj = geojson.Point((-3.68, 40.41, 25.14, 10.34))
>>> obj.is_valid
False
```

`errors` method provides collection of errors when validation GeoJSON objects.

```
>>> import geojson

>>> obj = geojson.Point((-3.68, 40.41, 25.14, 10.34))
>>> obj.errors()
'a position must have exactly 2 or 3 values'
```

4.6 generate_random

`geojson.utils.generate_random` yields a geometry type with random data


```
>>> import geojson

>>> geojson.utils.generate_random("LineString")
{"coordinates": [...], "type": "LineString"}

>>> geojson.utils.generate_random("Polygon")
{"coordinates": [...], "type": "Polygon"}
```


CHAPTER 5

Development

To build this project, run `python setup.py build`. To run the unit tests, run `python setup.py test`. To run the style checks, run `flake8` (install *flake8* if needed).

CHAPTER 6

Credits

- Sean Gillies <sgillies@frii.com>
- Matthew Russell <matt@sanoodi.com>
- Corey Farwell <coreyf@rwell.org>
- Blake Grotewold <hello@grotewold.me>
- Zsolt Ero <zsolt.ero@gmail.com>
- Sergey Romanov <xxsmotur@gmail.com>
- Ray Riga <ray@strongoutput.com>