

# Package ‘geojsonR’

July 22, 2025

**Type** Package

**Title** A GeoJson Processing Toolkit

**Version** 1.1.2

**Date** 2025-01-13

**BugReports** <https://github.com/mlampros/geojsonR/issues>

**URL** <https://github.com/mlampros/geojsonR>

**Description** Includes functions for processing GeoJson objects <<https://en.wikipedia.org/wiki/GeoJSON>> relying on 'RFC 7946' <<https://datatracker.ietf.org/doc/html/rfc7946>>. The geojson encoding is based on 'json11', a tiny JSON library for 'C++11' <<https://github.com/dropbox/json11>>. Furthermore, the source code is exported in R through the 'Rcpp' and 'RcppArmadillo' packages.

**License** MIT + file LICENSE

**Encoding** UTF-8

**Copyright** inst/COPYRIGHTS

**SystemRequirements** libarmadillo: apt-get install -y libarmadillo-dev (deb)

**Depends** R(>= 3.2.3)

**Imports** Rcpp (>= 0.12.9), R6

**LinkingTo** Rcpp, RcppArmadillo (>= 0.7.6)

**Suggests** testthat, covr, knitr, rmarkdown

**VignetteBuilder** knitr

**RoxygenNote** 7.3.2

**NeedsCompilation** yes

**Author** Lampros Mouselimis [aut, cre] (ORCID: <<https://orcid.org/0000-0002-8024-1546>>),  
Dropbox Inc [cph]

**Maintainer** Lampros Mouselimis <mouselimislampros@gmail.com>

**Repository** CRAN

**Date/Publication** 2025-01-13 19:10:05 UTC

## Contents

Dump_From_GeoJson . . . . .	2
Features_2Collection . . . . .	3
FROM_GeoJson . . . . .	4
FROM_GeoJson_Schema . . . . .	5
merge_files . . . . .	6
save_R_list_Features_2_FeatureCollection . . . . .	7
shiny_from_JSON . . . . .	9
TO_GeoJson . . . . .	10

<b>Index</b>	<b>17</b>
--------------	-----------

---

Dump_From_GeoJson	<i>returns a json-dump from a geojson file</i>
-------------------	--

---

### Description

returns a json-dump from a geojson file

### Usage

```
Dump_From_GeoJson(url_file)
```

### Arguments

url_file	either a string specifying the input path to a file OR a valid url (beginning with 'http.')
----------	---

### Value

a character string (json dump)

### Examples

```
## Not run:

library(geojsonR)

res = Dump_From_GeoJson("/myfolder/point.geojson")

## End(Not run)
```

---

Features\_2Collection *creates a FeatureCollection dump from multiple Feature geojson objects*

---

### Description

creates a FeatureCollection dump from multiple Feature geojson objects

### Usage

```
Features_2Collection(  
  Features_files_vec,  
  bbox_vec = NULL,  
  write_path = NULL,  
  verbose = FALSE  
)
```

### Arguments

Features_files_vec	a character vector specifying paths to files (Feature geojson objects)
bbox_vec	either NULL or a numeric vector
write_path	either NULL or a character string specifying a valid path to a file ( preferably with a <i>.geojson extension</i> ) where the output data will be saved
verbose	a boolean. If TRUE then information will be printed out in the console

### Details

The *Features\_2Collection* function utilizes internally a for-loop. In case of an error set the *verbose* parameter to TRUE to find out which file leads to this error.

### Value

a FeatureCollection dump

### Examples

```
## Not run:  
  
library(geojsonR)  
  
vec_files = c("/myfolder/Feature1.geojson", "/myfolder/Feature2.geojson",  
             "/myfolder/Feature3.geojson", "/myfolder/Feature4.geojson",  
             "/myfolder/Feature5.geojson")  
  
res = Features_2Collection(vec_files, bbox_vec = NULL)  
  
## End(Not run)
```

---

FROM_GeoJson	<i>reads GeoJson data</i>
--------------	---------------------------

---

**Description**

reads GeoJson data

**Usage**

```
FROM_GeoJson(
  url_file_string,
  Flatten_Coords = FALSE,
  Average_Coordinates = FALSE,
  To_List = FALSE
)
```

**Arguments**

<code>url_file_string</code>	a string specifying the input path to a file OR a geojson object (in form of a character string) OR a valid url (beginning with 'http..') pointing to a geojson object
<code>Flatten_Coords</code>	either TRUE or FALSE. If TRUE then the properties member of the geojson file will be omitted during parsing.
<code>Average_Coordinates</code>	either TRUE or FALSE. If TRUE then additionally a geojson-dump and the average latitude and longitude of the geometry object will be returned.
<code>To_List</code>	either TRUE or FALSE. If TRUE then the <i>coordinates</i> of the geometry object will be returned in form of a list, otherwise in form of a numeric matrix.

**Details**

The *FROM\_GeoJson* function is based on the 'RFC 7946' specification. Thus, geojson files/strings which include property-names other than the 'RFC 7946' specifies will return an error. To avoid errors of that kind a user should take advantage of the *FROM\_GeoJson\_Schema* function, which is not as strict concerning the property names.

**Value**

a (nested) list

**Examples**

```
## Not run:
library(geojsonR)
```

```
# INPUT IS A FILE

res = FROM_GeoJson(url_file_string = "/myfolder/feature_collection.geojson")

# INPUT IS A GEOJSON (character string)

tmp_str = '{ "type": "MultiPolygon", "coordinates": [
  [[102.0, 2.0], [103.0, 2.0], [103.0, 3.0], [102.0, 3.0], [102.0, 2.0]],
  [[100.0, 0.0], [101.0, 0.0], [101.0, 1.0], [100.0, 1.0], [100.0, 0.0]],
  [[100.2, 0.2], [100.8, 0.2], [100.8, 0.8], [100.2, 0.8], [100.2, 0.2]]
]
}'

res = FROM_GeoJson(url_file_string = tmp_str)

# INPUT IS A URL

res = FROM_GeoJson(url_file_string = "http://www.EXAMPLE_web_page.geojson")

## End(Not run)
```

---

FROM\_GeoJson\_Schema    *reads GeoJson data using a one-word-schema*

---

## Description

reads GeoJson data using a one-word-schema

## Usage

```
FROM_GeoJson_Schema(
  url_file_string,
  geometry_name = "",
  Average_Coordinates = FALSE,
  To_List = FALSE
)
```

## Arguments

`url_file_string` a string specifying the input path to a file OR a geojson object (in form of a character string) OR a valid url (beginning with 'http.>') pointing to a geojson object

`geometry_name` a string specifying the geometry name in the geojson string/file. The *geometry\_name* functions as a one-word schema and can significantly speed up the parsing of the data.

Average_Coordinates	either TRUE or FALSE. If TRUE then additionally a geojson-dump and the average latitude and longitude of the geometry object will be returned.
To_List	either TRUE or FALSE. If TRUE then the <i>coordinates</i> of the geometry object will be returned in form of a list, otherwise in form of a numeric matrix.

### Details

This function is appropriate when the property-names do not match exactly the 'RFC 7946' specification ( for instance if the *geometry* object-name appears as *location* as is the case sometimes in mongodb queries ). The user can then specify the *geometry\_name* as it exactly appears in the .geojson string/file (consult the example for more details). If no *geometry\_name* is given then recursion will be used, which increases the processing time. In case that the input .geojson object is of *type : Point, LineString, MultiPoint, Polygon, GeometryCollection, MultiLineString, MultiPolygon, Feature or FeatureCollection* with a second attribute name : *coordinates*, then the *geometry\_name* parameter is not necessary.

### Value

a (nested) list

### Examples

```
library(geojsonR)

# INPUT IS A GEOJSON (character string)

tmp_str = '{
  "name" : "example_name",
  "location" : {
    "type" : "Point",
    "coordinates" : [ -120.24, 39.21 ]
  }
}'

res = FROM_GeoJson_Schema(url_file_string = tmp_str, geometry_name = "location")
```

---

merge\_files

*merge json files (or any kind of text files) from a directory*

---

### Description

merge json files (or any kind of text files) from a directory

**Usage**

```
merge_files(
  INPUT_FOLDER,
  OUTPUT_FILE,
  CONCAT_DELIMITER = "\n",
  verbose = FALSE
)
```

**Arguments**

`INPUT_FOLDER` a character string specifying a path to the input folder

`OUTPUT_FILE` a character string specifying a path to the output file

`CONCAT_DELIMITER` a character string specifying the delimiter to use when merging the files

`verbose` either TRUE or FALSE. If TRUE then information will be printed in the console.

**Details**

This function is meant for json files but it can be applied to any kind of text files. It takes an input folder (*INPUT\_FOLDER*) and an output file (*OUTPUT\_FILE*) and merges all files from the *INPUT\_FOLDER* to a single *OUTPUT\_FILE* using the concatenation delimiter (*CONCAT\_DELIMITER*).

**Examples**

```
## Not run:
library(geojsonR)

merge_files(INPUT_FOLDER = "/my_folder/", OUTPUT_FILE = "output_file.json")

## End(Not run)
```

---

save\_R\_list\_Features\_2\_FeatureCollection

*creates a FeatureCollection from R list objects ( see the details section about the limitations of this function )*

---

**Description**

creates a FeatureCollection from R list objects ( see the details section about the limitations of this function )

**Usage**

```
save_R_list_Features_2_FeatureCollection(
  input_list,
  path_to_file = "",
  verbose = FALSE
)
```

**Arguments**

input_list	a list object that includes 1 or more geojson R list Features
path_to_file	either an empty string ("") or a valid path to a file where the output FeatureCollection will be saved
verbose	a boolean. If TRUE then information will be printed out in the console

**Details**

- it allows the following attributes: *'type'*, *'id'*, *'properties'* and *'geometry'*
- it allows only coordinates of type *'Polygon'* or *'MultiPolygon'* to be processed. In case of a *'Polygon'* there are 2 cases: (a.) Polygon WITHOUT interior rings (a numeric matrix is expected) and (b.) Polygon WITH interior rings (a list of numeric matrices is expected). See the test-cases if you receive an error for the correct format of the input data. In case of a *'MultiPolygon'* both Polygons with OR without interior rings can be included. Multipolygons are of the form: list of lists where each SUBLIST can be either a numeric matrix (Polygon without interior rings) or a list (Polygon with interior rings)
- the *properties* attribute must be a list that can take only *character strings*, *numeric* and *integer* values of SIZE 1. In case that any of the input properties is of SIZE > 1 then it will throw an error.

The *input\_list* parameter can be EITHER created from scratch OR GeoJson Features (in form of a FeatureCollection) can be loaded in R and modified so that this list can be processed by this function

**Value**

a FeatureCollection in form of a character string  
 a FeatureCollection saved in a file

**Examples**

```
## Not run:

library(geojsonR)

#-----
# valid example that will save the data to a file
#-----

Feature1 = list(type = "Feature",
               id = 1L,
               properties = list(prop1 = 'id', prop2 = 1.0234),
               geometry = list(type = 'Polygon',
                              coordinates = matrix(runif(20), nrow = 10, ncol = 2)))

Feature2 = list(type = "Feature",
               id = 2L,
               properties = list(prop1 = 'non-id', prop2 = 6.0987),
               geometry = list(type = 'MultiPolygon',
                              coordinates = list(matrix(runif(20), nrow = 10, ncol = 2),
```



```

matrix(runif(20), nrow = 10, ncol = 2)))

list_features = list(Feature1, Feature2)

path_feat_col = tempfile(fileext = '.geojson')

res = save_R_list_Features_2_FeatureCollection(input_list = list_features,
                                              path_to_file = path_feat_col,
                                              verbose = TRUE)

#-----
# validate that the file can be loaded
#-----

res_load = FROM_GeoJson_Schema(url_file_string = path_feat_col)
str(res_load)

#-----
# INVALID data types such as NA's will throw an ERROR
#-----

Feature1 = list(type = "Feature",
               id = 1L,
               properties = list(prop1 = NA, prop2 = 1.0234),
               geometry = list(type = 'Polygon',
                              coordinates = matrix(runif(20), nrow = 10, ncol = 2)))

list_features = list(Feature1, Feature2)

path_feat_col = tempfile(fileext = '.geojson')

res = save_R_list_Features_2_FeatureCollection(input_list = list_features,
                                              path_to_file = path_feat_col,
                                              verbose = TRUE)

## End(Not run)

```

---

shiny\_from\_JSON

*secondary function for shiny Applications*


---

### Description

secondary function for shiny Applications

### Usage

```
shiny_from_JSON(input_file)
```

**Arguments**

`input_file` a character string specifying a path to a file

**Details**

This function is meant for *shiny Applications*. To read a GeoJson file use either the *FROM\_GeoJson* or *FROM\_GeoJson\_Schema* function.

**Value**

a (nested) list

---

TO_GeoJson	<i>converts data to a GeoJson object</i>
------------	--

---

**Description**

converts data to a GeoJson object

converts data to a GeoJson object

**Usage**

```
# utl <- TO_GeoJson$new()
```

**Value**

a List

**Methods**

TO\_GeoJson\$new()  
 \_\_\_\_\_

Point(data, stringify = FALSE)  
 \_\_\_\_\_

MultiPoint(data, stringify = FALSE)  
 \_\_\_\_\_

LineString(data, stringify = FALSE)  
 \_\_\_\_\_

MultiLineString(data, stringify = FALSE)  
 \_\_\_\_\_

Polygon(data, stringify = FALSE)  
 \_\_\_\_\_

MultiPolygon(data, stringify = FALSE)

---

```
GeometryCollection(data, stringify = FALSE)
```

---

```
Feature(data, stringify = FALSE)
```

---

```
FeatureCollection(data, stringify = FALSE)
```

---

## Methods

### Public methods:

- [TO\\_GeoJson\\$new\(\)](#)
- [TO\\_GeoJson\\$Point\(\)](#)
- [TO\\_GeoJson\\$MultiPoint\(\)](#)
- [TO\\_GeoJson\\$LineString\(\)](#)
- [TO\\_GeoJson\\$MultiLineString\(\)](#)
- [TO\\_GeoJson\\$Polygon\(\)](#)
- [TO\\_GeoJson\\$MultiPolygon\(\)](#)
- [TO\\_GeoJson\\$GeometryCollection\(\)](#)
- [TO\\_GeoJson\\$Feature\(\)](#)
- [TO\\_GeoJson\\$FeatureCollection\(\)](#)
- [TO\\_GeoJson\\$clone\(\)](#)

### Method `new()`:

*Usage:*

```
TO_GeoJson$new()
```

### Method `Point()`:

*Usage:*

```
TO_GeoJson$Point(data, stringify = FALSE)
```

*Arguments:*

`data` a list specifying the geojson geometry object

`stringify` either TRUE or FALSE, specifying if the output should also include a geojson-dump (as a character string)

### Method `MultiPoint()`:

*Usage:*

```
TO_GeoJson$MultiPoint(data, stringify = FALSE)
```

*Arguments:*

`data` a list specifying the geojson geometry object

`stringify` either TRUE or FALSE, specifying if the output should also include a geojson-dump (as a character string)

**Method** LineString():*Usage:*

TO\_GeoJson\$LineString(data, stringify = FALSE)

*Arguments:*

data a list specifying the geojson geometry object

stringify either TRUE or FALSE, specifying if the output should also include a geojson-dump (as a character string)

**Method** MultiLineString():*Usage:*

TO\_GeoJson\$MultiLineString(data, stringify = FALSE)

*Arguments:*

data a list specifying the geojson geometry object

stringify either TRUE or FALSE, specifying if the output should also include a geojson-dump (as a character string)

**Method** Polygon():*Usage:*

TO\_GeoJson\$Polygon(data, stringify = FALSE)

*Arguments:*

data a list specifying the geojson geometry object

stringify either TRUE or FALSE, specifying if the output should also include a geojson-dump (as a character string)

**Method** MultiPolygon():*Usage:*

TO\_GeoJson\$MultiPolygon(data, stringify = FALSE)

*Arguments:*

data a list specifying the geojson geometry object

stringify either TRUE or FALSE, specifying if the output should also include a geojson-dump (as a character string)

**Method** GeometryCollection():*Usage:*

TO\_GeoJson\$GeometryCollection(data, stringify = FALSE)

*Arguments:*

data a list specifying the geojson geometry object

stringify either TRUE or FALSE, specifying if the output should also include a geojson-dump (as a character string)

**Method** Feature():*Usage:*

TO\_GeoJson\$Feature(data, stringify = FALSE)

*Arguments:*

`data` a list specifying the geojson geometry object  
`stringify` either TRUE or FALSE, specifying if the output should also include a geojson-dump (as a character string)

**Method** FeatureCollection():*Usage:*

```
TO_GeoJson$FeatureCollection(data, stringify = FALSE)
```

*Arguments:*

`data` a list specifying the geojson geometry object  
`stringify` either TRUE or FALSE, specifying if the output should also include a geojson-dump (as a character string)

**Method** clone(): The objects of this class are cloneable with this method.*Usage:*

```
TO_GeoJson$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

**Examples**

```
library(geojsonR)

# initialize class

init = TO_GeoJson$new()

# Examples covering all geometry-objects

# Point

point_dat = c(100, 1.01)

point = init$Point(point_dat, stringify = TRUE)
point

# MultiPoint

multi_point_dat = list(c(100, 1.01), c(200, 2.01))

multi_point = init$MultiPoint(multi_point_dat, stringify = TRUE)
multi_point

# LineString
```

```
linestring_dat = list(c(100, 1.01), c(200, 2.01))

line_string = init$LineString(linestring_dat, stringify = TRUE)
line_string

# MultiLineString

multilinestring_dat = list(list(c(100, 0.0), c(101, 1.0)), list(c(102, 2.0), c(103, 3.0)))

multiline_string = init$MultiLineString(multilinestring_dat, stringify = TRUE)
multiline_string

# Polygon (WITHOUT interior rings)

polygon_WITHOUT_dat = list(list(c(100, 1.01), c(200, 2.01), c(100, 1.0), c(100, 1.01)))

polygon_without = init$Polygon(polygon_WITHOUT_dat, stringify = TRUE)
polygon_without

# Polygon (WITH interior rings)

polygon_WITH_dat = list(list(c(100, 1.01), c(200, 2.01), c(100, 1.0), c(100, 1.01)),
                          list(c(50, 0.5), c(50, 0.8), c(50, 0.9), c(50, 0.5)))

polygon_with = init$Polygon(polygon_WITH_dat, stringify = TRUE)
polygon_with

# MultiPolygon

# the first polygon is without interior rings and the second one is with interior rings
multi_polygon_dat = list(list(list(c(102, 2.0), c(103, 2.0), c(103, 3.0), c(102, 2.0))),
                          list(list(c(100, 0.0), c(101, 1.0), c(101, 1.0), c(100, 0.0)),
                                list(c(100.2, 0.2), c(100.2, 0.8), c(100.8, 0.8), c(100.2, 0.2))))

multi_polygon = init$MultiPolygon(multi_polygon_dat, stringify = TRUE)
multi_polygon

# GeometryCollection (named list)

Point = c(100, 1.01)
```



```
prop1 = 0.0, vec = c(1,2,3), lst = list(a = 1, d = 2)))

feature_obj = init$Feature(feature_dat2, stringify = TRUE)
feature_obj
cat(feature_obj$json_dump)

# FeatureCollection (named list)

# takes as input the previously created 'feature_dat1', 'feature_dat2'
feature_col_dat = list(bbox = c(-10.01, -10.01, 10.01, 10.01),
                       features = list(Feature = feature_dat1, Feature = feature_dat2))
feature_col_dat

feature_collection_obj = init$FeatureCollection(feature_col_dat, stringify = TRUE)
feature_collection_obj
cat(feature_collection_obj$json_dump)
```



# Index

Dump\_From\_GeoJson, [2](#)

Features\_2Collection, [3](#)

FROM\_GeoJson, [4](#)

FROM\_GeoJson\_Schema, [5](#)

merge\_files, [6](#)

save\_R\_list\_Features\_2\_FeatureCollection,

[7](#)

shiny\_from\_JSON, [9](#)

TO\_GeoJson, [10](#)