Package 'wellknown'

January 10, 2020

Title Convert Between 'WKT' and 'GeoJSON'

Description Convert 'WKT' to 'GeoJSON' and 'GeoJSON' to 'WKT'. Functions included for converting between 'GeoJSON' to 'WKT', creating both 'GeoJSON' features, and non-features, creating 'WKT' from R objects (e.g., lists, data.frames, vectors), and linting 'WKT'.

Version 0.6.0

License MIT + file LICENSE

LazyData true

URL `https://docs.ropensci.org/wellknown`,
`https://github.com/ropensci/wellknown`

BugReports `https://github.com/ropensci/wellknown/issues`

VignetteBuilder knitr

Imports jsonlite, V8 (>= 1.0.2)

Suggests knitr, rmarkdown, leaflet (>= 1.0.0), testthat

RoxygenNote 7.0.2

X-schema.org-applicationCategory Geosptial

X-schema.org-keywords spatial, geospatial, wkt, wkb, well-known-text, geojson, binary, conversion

X-schema.org-isPartOf https://ropensci.org

NeedsCompilation no

Author Scott Chamberlain [aut, cre] (<https://orcid.org/0000-0003-1444-9135>)

Maintainer Scott Chamberlain <`myrmecocystus@gmail.com`>

Repository CRAN

Date/Publication 2020-01-10 05:30:02 UTC

R topics documented:

`wellknown-package` *WKT to GeoJSON and vice versa*

Description

WKT to GeoJSON and vice versa

Author(s)

Scott Chamberlain <`myrmecocystus@gmail.com`>

Examples

```
# GeoJSON to WKT
point <- list(Point = c(116.4, 45.2, 11.1))
geojson2wkt(point)

# WKT to GeoJSON
str <- "POINT (-116.4000000000000057 45.2000000000000028)"
wkt2geojson(str)

## lint WKT
lint("POINT (1 2)")
lint("POINT (1 2 3 4 5)")
```

```
# WKT <--> WKB
wkt_wkb("POINT (-116.4 45.2)")
wkb_wkt(wkt_wkb("POINT (-116.4 45.2)"))
```

`as_featurecollection` *As featurecollection*

Description

Helper function to make a FeatureCollection list object for use in vizualizing, e.g., with `leaflet`

Usage

```
as_featurecollection(x)
```

Arguments

`x`	(list) GeoJSON as a list

Examples

```
str <- 'MULTIPOINT ((100.000 3.101), (101.000 2.100), (3.140 2.180),
(31.140 6.180), (31.140 78.180))'
x <- wkt2geojson(str, fmt = 2)
as_featurecollection(x)
```

`as_json` *Convert geojson R list to JSON*

Description

Convert geojson R list to JSON

Usage

```
as_json(x, pretty = TRUE, auto_unbox = TRUE, ...)
```

Arguments

`x`	Output from `wkt2geojson()`
`pretty`	(logical) Adds indentation whitespace to JSON output. Can be `TRUE`/`FALSE` or a number specifying the number of spaces to indent. See `jsonlite::prettify()`. Default: `TRUE`. Having `TRUE` as default makes it easy to copy paste to a text editor, etc.
`auto_unbox`	(logical) Automatically unbox all atomic vectors of length 1. Default: `TRUE`
`...`	Further args passed on to `jsonlite::toJSON()`

Examples

```
str <- "POLYGON ((100 0.1, 101.1 0.3, 101 0.5, 100 0.1),
   (103.2 0.2, 104.8 0.2, 100.8 0.8, 103.2 0.2))"
as_json(wkt2geojson(str))
as_json(wkt2geojson(str), FALSE)
```

`circularstring` *Make WKT circularstring objects*

Description

Make WKT circularstring objects

Usage

```
circularstring(..., fmt = 16)
```

Arguments

`...`	A GeoJSON-like object representing a Point, LineString, Polygon, MultiPolygon, etc.
`fmt`	Format string which indicates the number of digits to display after the decimal point when formatting coordinates. Max: 20

See Also

Other R-objects: `geometrycollection()`, `linestring()`, `multilinestring()`, `multipoint()`, `multipolygon()`, `point()`, `polygon()`

Examples

```
## empty circularstring
circularstring("empty")
# circularstring("stuff")

# Character string
circularstring("CIRCULARSTRING(1 5, 6 2, 7 3)")

# data.frame
df <- data.frame(lon = c(-116.4, -118), lat = c(45.2, 47))
circularstring(df, fmt=1)
df <- data.frame(lon=c(-116.4, -118, -120), lat=c(45.2, 47, 49))
circularstring(df, fmt=1)

# matrix
mat <- matrix(c(-116.4,-118, 45.2, 47), ncol = 2)
circularstring(mat, fmt=1)
mat2 <- matrix(c(-116.4, -118, -120, 45.2, 47, 49), ncol = 2)
circularstring(mat2, fmt=1)
```

```
# list
x <- list(c(1, 5), c(6, 2), c(7, 3))
circularstring(x, fmt=2)
```

---

geojson2wkt *Convert GeoJSON-like objects to WKT*

Description

Convert GeoJSON-like objects to WKT

Usage

```
geojson2wkt(obj, fmt = 16, third = "z", ...)
```

Arguments

`obj`	(list/json/character) A GeoJSON-like object representing a Point, MultiPoint, LineString, MultiLineString, Polygon, MultiPolygon, or GeometryCollection
`fmt`	Format string which indicates the number of digits to display after the decimal point when formatting coordinates. Max: 20
`third`	(character) Only applicable when there are three dimensions. If `m`, assign a `M` value for a measurement, and if `z` assign a `Z` value for three-dimenionsal system. Case is ignored. An `M` value represents a measurement, while a `Z` value usually represents altitude (but can be something like depth in a water based location).
`...`	Further args passed on to `jsonlite::fromJSON()` only in the event of json passed as a character string (can also be json of class `json` as returned from `jsonlite::toJSON()` or simply coerced to `json` by adding the class manually)

Inputs

Input to `obj` parameter can take two forms:

- A list with named elements `type` and `coordinates` OR `type` and `geometries` (only in the case of GeometryCollection). e.g., `list(type = "Point",coordinates = c(1,0))`
- A list with single named element in the set Point, Multipoint, Polygon, Multipolygon, Linestring, Multilinestring,or Geometrycollection, e.g., `list(Point = c(1,0))` - Note that this format is not proper GeoJSON, but is less verbose than the previous format, so should save the user time and make it easier to use.

Each point

For any one point, 2 to 4 values can be used:

- 2 values: longitude, latitude
- 3 values: longitude, latitude, altitude
- 4 values: longitude, latitude, altitude, measure

The 3rd value is typically altitude though can be depth in an aquatic context.

The 4th value is a measurement of some kind.

The GeoJSON spec `https://tools.ietf.org/html/rfc7946` actually doesn't allow a 4th value for a point, but we allow it here since we're converting to WKT which does allow a 4th value for a point.

Coordinates data formats

Coordinates data should follow the following formats:

- Point: a vector or list, with a single point (2-4 values)
- MultiPoint: a matrix, with N points
- Linestring: a matrix, with N points
- MultiLinestring: the top most level is a list, containing N matrices
- Polygon: the top most level is a list, containing N matrices
- MultiPolygon: the top most level is a list, the next level is N lists, each of them containing N matrices
- Geometrycollection: a list containing any combination and number of the above types

Matrices by definition can not have unequal lengths in their columns, so we don't have to check for that user error.

Each matrix can have any number of rows, and from 2 to 4 columns. If > 5 columns we stop with an error message.

References

`https://tools.ietf.org/html/rfc7946`, `https://en.wikipedia.org/wiki/Well-known_text`

See Also

`wkt2geojson()`

Examples

```
# point
## new format
point <- list(Point = c(116.4, 45.2))
geojson2wkt(point)
## old format, warns
point <- list(type = 'Point', coordinates = c(116.4, 45.2))
```

```
geojson2wkt(point)

# multipoint
## new format
mp <- list(MultiPoint = matrix(c(100, 101, 3.14, 3.101, 2.1, 2.18),
   ncol = 2))
geojson2wkt(mp)
## 3D
mp <- list(MultiPoint = matrix(c(100, 101, 3, 3, 2, 2, 4, 5, 6),
   ncol = 3))
geojson2wkt(mp)
## old format, warns
mp <- list(
  type = 'MultiPoint',
  coordinates = matrix(c(100, 101, 3.14, 3.101, 2.1, 2.18), ncol = 2)
)
geojson2wkt(mp)

# linestring
## new format
st <- list(LineString = matrix(c(0.0, 2.0, 4.0, 5.0,
                               0.0, 1.0, 2.0, 4.0),
                               ncol = 2))
geojson2wkt(st)
## 3D
st <- list(LineString = matrix(
  c(0.0, 0, 0,
   2, 1, 5,
   100, 300, 800), nrow = 3))
geojson2wkt(st, fmt = 2)
geojson2wkt(st, fmt = 2, third = "m")
## old format, warns
st <- list(
  type = 'LineString',
  coordinates = matrix(c(0.0, 2.0, 4.0, 5.0,
                         0.0, 1.0, 2.0, 4.0), ncol = 2)
)
geojson2wkt(st)
## 3D
st <- list(LineString = matrix(c(0.0, 2.0, 4.0, 5.0,
                               0.0, 1.0, 2.0, 4.0,
                               10, 20, 30, 40),
                               ncol = 3))
geojson2wkt(st, fmt = 2)

## 4D
st <- list(LineString = matrix(c(0.0, 2.0, 4.0, 5.0,
                               0.0, 1.0, 2.0, 4.0,
                               10, 20, 30, 40,
                               1, 2, 3, 4),
                               ncol = 4))
geojson2wkt(st, fmt = 2)
```

```
# multilinestring
## new format
multist <- list(MultiLineString = list(
   matrix(c(0, -2, -4, -1, -3, -5), ncol = 2),
   matrix(c(1.66, 10.9999, 10.9, 0, -31.5, 3.0, 1.1, 0), ncol = 2)
 )
)
geojson2wkt(multist)
## 3D
multist <- list(MultiLineString = list(
   matrix(c(0, -2, -4, -1, -3, -5, 100, 200, 300), ncol = 3),
   matrix(c(1, 10, 10.9, 0, -31.5, 3.0, 1.1, 0, 3, 3, 3, 3), ncol = 3)
 )
)
geojson2wkt(multist, fmt = 2)
geojson2wkt(multist, fmt = 2, third = "m")
## old format, warns
multist <- list(
  type = 'MultiLineString',
  coordinates = list(
   matrix(c(0, -2, -4, -1, -3, -5), ncol = 2),
   matrix(c(1.66, 10.9999, 10.9, 0, -31.5, 3.0, 1.1, 0), ncol = 2)
 )
)
geojson2wkt(multist)

## points within MultiLineString that differ
## -> use length of longest
## -> fill with zeros
# 3D and 2D
multist <- list(MultiLineString = list(
    matrix(1:6, ncol = 3), matrix(1:8, ncol = 2)))
geojson2wkt(multist, fmt = 0)
# 4D and 2D
multist <- list(MultiLineString = list(
    matrix(1:8, ncol = 4), matrix(1:8, ncol = 2)))
geojson2wkt(multist, fmt = 0)
# 2D and 2D
multist <- list(MultiLineString = list(
    matrix(1:4, ncol = 2), matrix(1:8, ncol = 2)))
geojson2wkt(multist, fmt = 0)
# 5D and 2D - FAILS
# multist <- list(MultiLineString = list(
#    matrix(1:10, ncol = 5), matrix(1:8, ncol = 2)))
# geojson2wkt(multist, fmt = 0)


# polygon
## new format
poly <- list(Polygon = list(
   matrix(c(100.001, 101.1, 101.001, 100.001, 0.001, 0.001, 1.001, 0.001), ncol = 2),
   matrix(c(100.201, 100.801, 100.801, 100.201, 0.201, 0.201, 0.801, 0.201), ncol = 2)
```

```
))
geojson2wkt(poly)
geojson2wkt(poly, fmt=6)
## 3D
poly <- list(Polygon = list(
   matrix(c(100.1, 101.1, 101.1, 100.1, 0.1, 0.1, 1.1, 0.1, 1, 1, 1, 1), ncol = 3),
   matrix(c(100.2, 100.8, 100.8, 100.2, 0.2, 0.2, 0.80, 0.2, 3, 3, 3, 3), ncol = 3)
))
geojson2wkt(poly, fmt = 2)
geojson2wkt(poly, fmt = 2, third = "m")
## old format, warns
poly <- list(
  type = 'Polygon',
  coordinates = list(
    matrix(c(100.001, 101.1, 101.001, 100.001, 0.001, 0.001, 1.001, 0.001),
      ncol = 2),
    matrix(c(100.201, 100.801, 100.801, 100.201, 0.201, 0.201, 0.801, 0.201),
      ncol = 2)
  )
)
geojson2wkt(poly)
geojson2wkt(poly, fmt=6)

## points within Polygon that differ
## -> use length of longest
## -> fill with zeros
# 3D and 2D
poly <- list(Polygon = list(
   matrix(c(100, 101, 101, 100, 0.1, 0.2, 0.3, 0.1, 5, 6, 7, 8), ncol = 3),
   matrix(c(40, 41, 61, 40, 0.1, 0.2, 0.3, 0.1), ncol = 2)
))
geojson2wkt(poly, fmt = 0)
# 4D and 2D
poly <- list(Polygon = list(
   matrix(c(100, 101, 101, 100, 0.1, 0.2, 0.3, 0.1, 5, 6, 7, 8, 1, 1, 1, 1),
     ncol = 4),
   matrix(c(40, 41, 61, 40, 0.1, 0.2, 0.3, 0.1), ncol = 2)
))
geojson2wkt(poly, fmt = 0)
# 5D and 2D - FAILS
# multist <- list(Polygon = list(
#    matrix(1:10, ncol = 5), matrix(1:8, ncol = 2)))
# geojson2wkt(poly, fmt = 0)



# multipolygon
## new format
mpoly <- list(MultiPolygon = list(
  list(
    matrix(c(100, 101, 101, 100, 0.001, 0.001, 1.001, 0.001), ncol = 2),
    matrix(c(100.2, 100.8, 100.8, 100.2, 0.2, 0.2, 0.8, 0.2), ncol = 2)
  ),
  list(
    matrix(c(1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 1.0), ncol = 2),
    matrix(c(9.0, 10.0, 11.0, 12.0, 1.0, 2.0, 3.0, 4.0, 9.0), ncol = 2)
  )
))
geojson2wkt(mpoly, fmt=2)
## 3D
mpoly <- list(MultiPolygon = list(
  list(
    matrix(c(100, 101, 101, 100, 0.001, 0.001, 1.001, 0.001, 1, 1, 1, 1),
      ncol = 3),
    matrix(c(100.2, 100.8, 100.8, 100.2, 0.2, 0.2, 0.8, 0.2, 3, 4, 5, 6),
      ncol = 3)
  ),
  list(
    matrix(c(1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 1.0, 1, 1, 1, 1),
      ncol = 3),
    matrix(c(9.0, 10.0, 11.0, 12.0, 1.0, 2.0, 3.0, 4.0, 9.0, 9, 9, 9, 9),
      ncol = 3)
  )
))
geojson2wkt(mpoly, fmt=2)
geojson2wkt(mpoly, fmt=2, third = "m")
## old format, warns
mpoly <- list(
  type = 'MultiPolygon',
  coordinates = list(
    list(
      matrix(c(100, 101, 101, 100, 0.001, 0.001, 1.001, 0.001), ncol = 2),
      matrix(c(100.2, 100.8, 100.8, 100.2, 0.2, 0.2, 0.8, 0.2), ncol = 2)
    ),
    list(
      matrix(c(1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 1.0), ncol = 3),
      matrix(c(9.0, 10.0, 11.0, 12.0, 1.0, 2.0, 3.0, 4.0, 9.0), ncol = 3)
    )
  )
)
geojson2wkt(mpoly, fmt=2)

mpoly2 <- list(
  type = "MultiPolygon",
  coordinates = list(
    list(list(c(30, 20), c(45, 40), c(10, 40), c(30, 20))),
    list(list(c(15, 5), c(40, 10), c(10, 20), c(5 ,10), c(15, 5)))
  )
)

mpoly2 <- list(
  type = "MultiPolygon",
  coordinates = list(
    list(
      matrix(c(30, 45, 10, 30, 20, 40, 40, 20), ncol = 2)
    ),
    list(
      matrix(c(15, 40, 10, 5, 15, 5, 10, 20, 10, 5), ncol = 2)
    )
  )
)
geojson2wkt(mpoly2, fmt=1)

## points within MultiPolygon that differ
## -> use length of longest
## -> fill with zeros
# 3D and 2D
mpoly <- list(MultiPolygon = list(
  list(
    matrix(c(40, 130, 155, 40, 20, 34, 34, 20), ncol = 2),
    matrix(c(30, 40, 54, 30, 0.1, 42, 62, 0.1, 1, 1, 1, 1), ncol = 3)
  ),
  list(
    matrix(c(9, 49, 79, 9, 11, 35, 15, 11), ncol = 2),
    matrix(c(1, 33, 59, 1, 5, 16, 36, 5), ncol = 2)
  )
))
geojson2wkt(mpoly, fmt = 0)
# 4D and 2D
mpoly <- list(MultiPolygon = list(
  list(
    matrix(c(40, 130, 155, 40, 20, 34, 34, 20), ncol = 2),
    matrix(c(30, 40, 54, 30, 0.1, 42, 62, 0.1, 1, 1, 1, 1, 0, 0, 0, 0), ncol = 4)
  ),
  list(
    matrix(c(9, 49, 79, 9, 11, 35, 15, 11), ncol = 2),
    matrix(c(1, 33, 59, 1, 5, 16, 36, 5), ncol = 2)
  )
))
geojson2wkt(mpoly, fmt = 0)
# 5D and 2D - FAILS
mpoly <- list(MultiPolygon = list(
  list(
    matrix(c(40, 130, 155, 40, 20, 34, 34, 20), ncol = 2),
    matrix(c(30, 40, 54, 30,
             0.1, 42, 62, 0.1,
             1, 1, 1, 1,
             0, 0, 0, 0,
             0, 0, 0, 0), ncol = 5)
  ),
  list(
    matrix(c(9, 49, 79, 9, 11, 35, 15, 11), ncol = 2),
    matrix(c(1, 33, 59, 1, 5, 16, 36, 5), ncol = 2)
  )
))
# geojson2wkt(mpoly, fmt = 0)
# geometrycollection
## new format
gmcoll <- list(GeometryCollection = list(
 list(Point = c(0.0, 1.0)),
 list(LineString = matrix(c(0.0, 2.0, 4.0, 5.0,
                           0.0, 1.0, 2.0, 4.0),
                           ncol = 2)),
 list(Polygon = list(
   matrix(c(100.001, 101.1, 101.001, 100.001, 0.001, 0.001, 1.001, 0.001),
     ncol = 2),
   matrix(c(100.201, 100.801, 100.801, 100.201, 0.201, 0.201, 0.801, 0.201),
     ncol = 2)
  ))
))
geojson2wkt(gmcoll, fmt=0)
## old format, warns
gmcoll <- list(
 type = 'GeometryCollection',
 geometries = list(
   list(type = 'Point', coordinates = c(0.0, 1.0)),
   list(type = 'LineString', coordinates = matrix(c(0.0, 2.0, 4.0, 5.0,
                           0.0, 1.0, 2.0, 4.0),
                           ncol = 2)),
   list(type = 'Polygon', coordinates = list(
     matrix(c(100.001, 101.1, 101.001, 100.001, 0.001, 0.001, 1.001, 0.001),
       ncol = 2),
     matrix(c(100.201, 100.801, 100.801, 100.201, 0.201, 0.201, 0.801, 0.201),
       ncol = 2)
  ))
 )
)
geojson2wkt(gmcoll, fmt=0)

# Convert geojson as character string to WKT
# new format
str <- '{
   "Point": [
       -105.01621,
       39.57422
   ]
}'
geojson2wkt(str)

## old format, warns
str <- '{
   "type": "Point",
   "coordinates": [
       -105.01621,
       39.57422
   ]
}'
geojson2wkt(str)
```

```
## new format
str <- '{"LineString":[[0,0,10],[2,1,20],[4,2,30],[5,4,40]]}'
geojson2wkt(str)
## old format, warns
str <-
'{"type":"LineString","coordinates":[[0,0,10],[2,1,20],[4,2,30],[5,4,40]]}'
geojson2wkt(str)

# From a jsonlite json object
library("jsonlite")
json <- toJSON(list(Point=c(-105,39)), auto_unbox=TRUE)
geojson2wkt(json)
## old format, warns
json <- toJSON(list(type="Point", coordinates=c(-105,39)), auto_unbox=TRUE)
geojson2wkt(json)
```

`geometrycollection` *Make WKT geometrycollection objects*

Description

Make WKT geometrycollection objects

Usage

```
geometrycollection(...)
```

Arguments

`...`	Character string WKT objects representing a Point, LineString, Polygon, etc.

Details

This is different from the other functions that create WKT from R objects, in that we can't do the same thing for GeometryCollection's since many different WkT object could be created from the same input. So, this function accepts WKT strings already formed and attempts to creat a GeommetryCollection from them.

See Also

Other R-objects: `circularstring()`, `linestring()`, `multilinestring()`, `multipoint()`, `multipolygon()`, `point()`, `polygon()`

Examples

```
## empty geometrycollection
geometrycollection("empty")
# geometrycollection("stuff")
# Character string, returns itself
geometrycollection("GEOMETRYCOLLECTION(POINT(4 6), LINESTRING(4 6, 7 10))")

# From a point
geometrycollection(point(-116.4, 45.2))

# From two points
geometrycollection(point(-116.4, 45.2), point(-118.4, 49.2))

# From various object types
geometrycollection(point(-116.4, 45.2),
 linestring("LINESTRING (-116.4 45.2, -118.0 47.0)"),
 circularstring(list(c(1, 5), c(6, 2), c(7, 3)), fmt = 2)
)
```

get_centroid *Get a centroid from WKT or geojson*

Description

Get a centroid from WKT or geojson

Usage

```
get_centroid(x)
```

Arguments

x	Input, a wkt character string or geojson class object

Value

A length 2 numeric vector, with longitude first, latitude second

Examples

```
# WKT
str <- "POINT (-116.4000000000000057 45.2000000000000028)"
get_centroid(str)
str <- 'MULTIPOINT ((100.000 3.101), (101.000 2.100), (3.140 2.180))'
get_centroid(str)
str <- "MULTIPOLYGON (((40 40, 20 45, 45 30, 40 40)),
 ((20 35, 45 20, 30 5, 10 10, 10 30, 20 35), (30 20, 20 25, 20 15, 30 20)))"
get_centroid(str)

# Geojson as geojson class
str <- "POINT (-116.4000000000000057 45.2000000000000028)"
get_centroid(wkt2geojson(str))
str <- 'MULTIPOINT ((100.000 3.101), (101.000 2.100), (3.140 2.180))'
get_centroid(wkt2geojson(str))
```

```
str <- "MULTIPOLYGON (((40 40, 20 45, 45 30, 40 40)),
 ((20 35, 45 20, 30 5, 10 10, 10 30, 20 35), (30 20, 20 25, 20 15, 30 20)))"
get_centroid(wkt2geojson(str))
```

linestring *Make WKT linestring objects*

Description

Make WKT linestring objects

Usage

```
linestring(..., fmt = 16, third = "z")
```

Arguments

`...`	A GeoJSON-like object representing a Point, LineString, Polygon, MultiPolygon, etc.
`fmt`	Format string which indicates the number of digits to display after the decimal point when formatting coordinates. Max: 20
`third`	(character) Only applicable when there are three dimensions. If `m`, assign a `M` value for a measurement, and if `z` assign a `Z` value for three-dimenionsal system. Case is ignored. An `M` value represents a measurement, while a `Z` value usually represents altitude (but can be something like depth in a water based location).

See Also

Other R-objects: `circularstring()`, `geometrycollection()`, `multilinestring()`, `multipoint()`, `multipolygon()`, `point()`, `polygon()`

Examples

```
## empty linestring
linestring("empty")
# linestring("stuff")

## character string
linestring("LINESTRING (-116.4 45.2, -118.0 47.0)")

# numeric
## 2D
linestring(c(100.000, 0.000), c(101.000, 1.000), fmt=2)
linestring(c(100.0, 0.0), c(101.0, 1.0), c(120.0, 5.00), fmt=2)
## 3D
linestring(c(0.0, 0.0, 10.0), c(2.0, 1.0, 20.0),
           c(4.0, 2.0, 30.0), c(5.0, 4.0, 40.0), fmt=2)
## 4D
```

```
linestring(c(0.0, 0.0, 10.0, 5.0), c(2.0, 1.0, 20.0, 5.0),
           c(4.0, 2.0, 30.0, 5.0), c(5.0, 4.0, 40.0, 5.0), fmt=2)

# data.frame
df <- data.frame(lon=c(-116.4,-118), lat=c(45.2,47))
linestring(df, fmt=1)
df <- data.frame(lon=c(-116.4,-118,-120), lat=c(45.2,47,49))
linestring(df, fmt=1)
## 3D
df$altitude <- round(runif(NROW(df), 10, 50))
linestring(df, fmt=1)
linestring(df, fmt=1, third = "m")
## 4D
df$weight <- round(runif(NROW(df), 0, 1), 1)
linestring(df, fmt=1)


# matrix
mat <- matrix(c(-116.4,-118, 45.2, 47), ncol = 2)
linestring(mat, fmt=1)
mat2 <- matrix(c(-116.4, -118, -120, 45.2, 47, 49), ncol = 2)
linestring(mat2, fmt=1)
## 3D
mat <- matrix(c(df$long, df$lat, df$altitude), ncol = 3)
polygon(mat, fmt=2)
polygon(mat, fmt=2, third = "m")
## 4D
mat <- matrix(unname(unlist(df)), ncol = 4)
polygon(mat, fmt=2)

# list
linestring(list(c(100.000, 0.000), c(101.000, 1.000)), fmt=2)
## 3D
line <- list(c(100, 0, 1), c(101, 0, 1), c(101, 1, 1),
  c(100, 0, 1))
linestring(line, fmt=2)
linestring(line, fmt=2, third = "m")
## 4D
line <- list(c(100, 0, 1, 40), c(101, 0, 1, 44), c(101, 1, 1, 45),
  c(100, 0, 1, 49))
linestring(line, fmt=2)
```

`lint` *Validate WKT strings*

Description

Validate WKT strings

Usage

```
lint(str)
```

Arguments

`str`	A WKT string

Details

This function uses R regex - there's no error messages about what is wrong in the WKT.

Value

A logical (`TRUE` or `FALSE`)

Examples

```
lint("POINT (1 2)")
lint("POINT (1 2 3)")
lint("LINESTRING EMPTY")
lint("LINESTRING (100 0, 101 1)")
lint("MULTIPOINT EMPTY")
lint("MULTIPOINT ((1 2), (3 4))")
lint("MULTIPOINT ((1 2), (3 4), (-10 100))")
lint("POLYGON ((1 2, 3 4, 0 5, 1 2))")
lint("POLYGON((20.3 28.6, 20.3 19.6, 8.5 19.6, 8.5 28.6, 20.3 28.6))")
lint("MULTIPOLYGON (((30 20, 45 40, 10 40, 30 20)), ((15 5, 40 10, 10 20, 5 10, 15 5)))")
lint("TRIANGLE ((0 0, 0 1, 1 1, 0 0))")
lint("TRIANGLE ((0.1 0.1, 0.1 1.1, 1.1 1.1, 0.1 0.1))")
lint("CIRCULARSTRING (1 5, 6 2, 7 3)")
lint("CIRCULARSTRING (1 5, 6 2, 7 3, 5 6, 4 3)")
lint('COMPOUNDCURVE (CIRCULARSTRING (1 0, 0 1, -1 0), (-1 0, 2 0))')
```

`multilinestring` *Make WKT multilinestring objects*

Description

Make WKT multilinestring objects

Usage

```
multilinestring(..., fmt = 16, third = "z")
```

Arguments

`...`	A GeoJSON-like object representing a Point, LineString, Polygon, MultiPolygon, etc.
`fmt`	Format string which indicates the number of digits to display after the decimal point when formatting coordinates. Max: 20
`third`	(character) Only applicable when there are three dimensions. If `m`, assign a `M` value for a measurement, and if `z` assign a `Z` value for three-dimenionsal system. Case is ignored. An `M` value represents a measurement, while a `Z` value usually represents altitude (but can be something like depth in a water based location).

Details

There is no `numeric` input option for multilinestring. There is no way as of yet to make a nested multilinestring with `data.frame` input, but you can do so with list input. See examples.

See Also

Other R-objects: `circularstring()`, `geometrycollection()`, `linestring()`, `multipoint()`, `multipolygon()`, `point()`, `polygon()`

Examples

```
## empty multilinestring
multilinestring("empty")
# multilinestring("stuff")

# character string
x <- "MULTILINESTRING ((30 20, 45 40, 10 40), (15 5, 40 10, 10 20))"
multilinestring(x)

# data.frame
df <- data.frame(long = c(30, 45, 10), lat = c(20, 40, 40))
df2 <- data.frame(long = c(15, 40, 10), lat = c(5, 10, 20))
multilinestring(df, df2, fmt=0)
lint(multilinestring(df, df2, fmt=0))
wktview(multilinestring(df, df2), zoom=3)

# matrix
mat <- matrix(c(df$long, df$lat), ncol = 2)
mat2 <- matrix(c(df2$long, df2$lat), ncol = 2)
multilinestring(mat, mat2, fmt=0)

# list
x1 <- list(c(30, 20), c(45, 40), c(10, 40))
x2 <- list(c(15, 5), c(40, 10), c(10, 20))
multilinestring(x1, x2, fmt=2)

polys <- list(
  list(c(30, 20), c(45, 40), c(10, 40)),
  list(c(15, 5), c(40, 10), c(10, 20))
)
wktview(multilinestring(polys, fmt=2), zoom=3)

# 3D
## data.frame
df <- data.frame(long = c(30, 45, 10), lat = c(20, 40, 40), altitude = 1:3)
df2 <- data.frame(long = c(15, 40, 10), lat = c(5, 10, 20), altitude = 1:3)
multilinestring(df, df2, fmt=0)
multilinestring(df, df2, fmt=0, third = "m")
## matrix
mat <- matrix(unname(unlist(df)), ncol = 3)
mat2 <- matrix(unname(unlist(df2)), ncol = 3)
multilinestring(mat, mat2, fmt=0)
multilinestring(mat, mat2, fmt=0, third = "m")
## list
x1 <- list(c(30, 20, 1), c(45, 40, 1), c(10, 40, 1))
x2 <- list(c(15, 5, 0), c(40, 10, 3), c(10, 20, 4))
multilinestring(x1, x2, fmt=2)
multilinestring(x1, x2, fmt=2, third = "m")


# 4D
## data.frame
df <- data.frame(long = c(30, 45, 10), lat = c(20, 40, 40),
  altitude = 1:3, weight = 4:6)
df2 <- data.frame(long = c(15, 40, 10), lat = c(5, 10, 20),
  altitude = 1:3, weight = 4:6)
multilinestring(df, df2, fmt=0)
## matrix
mat <- matrix(unname(unlist(df)), ncol = 4)
mat2 <- matrix(unname(unlist(df2)), ncol = 4)
multilinestring(mat, mat2, fmt=0)
## list
x1 <- list(c(30, 20, 1, 40), c(45, 40, 1, 40), c(10, 40, 1, 40))
x2 <- list(c(15, 5, 0, 40), c(40, 10, 3, 40), c(10, 20, 4, 40))
multilinestring(x1, x2, fmt=2)
```

multipoint *Make WKT multipoint objects*

Description

Make WKT multipoint objects

Usage

```
multipoint(..., fmt = 16, third = "z")
```

Arguments

`...`	A GeoJSON-like object representing a Point, LineString, Polygon, MultiPolygon, etc.
`fmt`	Format string which indicates the number of digits to display after the decimal point when formatting coordinates. Max: 20
`third`	(character) Only applicable when there are three dimensions. If `m`, assign a `M` value for a measurement, and if `z` assign a `Z` value for three-dimenionsal system. Case is ignored. An `M` value represents a measurement, while a `Z` value usually represents altitude (but can be something like depth in a water based location).

See Also

Other R-objects: `circularstring()`, `geometrycollection()`, `linestring()`, `multilinestring()`, `multipolygon()`, `point()`, `polygon()`

Examples

```
## empty multipoint
multipoint("empty")
# multipoint("stuff")

# numeric
multipoint(c(100.000, 3.101), c(101.000, 2.100), c(3.140, 2.180))

# data.frame
df <- us_cities[1:25, c('long', 'lat')]
multipoint(df)

# matrix
mat <- matrix(c(df$long, df$lat), ncol = 2)
multipoint(mat)

# list
multipoint(list(c(100.000, 3.101), c(101.000, 2.100), c(3.140, 2.180)))


## a 3rd point is included
multipoint(c(100, 3, 0), c(101, 2, 0), c(3, 2, 0),
  third = "z", fmt = 1)
multipoint(c(100, 3, 0), c(101, 2, 0), c(3, 2, 0),
  third = "m", fmt = 1)

df <- us_cities[1:25, c('long', 'lat')]
df$altitude <- round(runif(25, 100, 500))
multipoint(df, fmt = 2)
multipoint(df, fmt = 2, third = "m")

mat <- matrix(1:9, 3)
multipoint(mat)
multipoint(mat, third = "m")
```

```
x <- list(c(100, 3, 0), c(101, 2, 1), c(3, 2, 5))
multipoint(x)


## a 4th point is included
multipoint(
  c(100, 3, 0, 500), c(101, 2, 0, 505), c(3, 2, 0, 600),
  fmt = 1)

df <- us_cities[1:25, c('long', 'lat')]
df$altitude <- round(runif(25, 100, 500))
df$weight <- round(runif(25, 1, 100))
multipoint(df, fmt = 2)

mat <- matrix(1:12, 3)
multipoint(mat)

x <- list(c(100, 3, 0, 300), c(101, 2, 1, 200), c(3, 2, 5, 100))
multipoint(x, fmt = 3)
```

multipolygon *Make WKT multipolygon objects*

Description

Make WKT multipolygon objects

Usage

```
multipolygon(..., fmt = 16, third = "z")
```

Arguments

`...`	A GeoJSON-like object representing a Point, LineString, Polygon, MultiPolygon, etc.
`fmt`	Format string which indicates the number of digits to display after the decimal point when formatting coordinates. Max: 20
`third`	(character) Only applicable when there are three dimensions. If `m`, assign a `M` value for a measurement, and if `z` assign a `Z` value for three-dimenionsal system. Case is ignored. An `M` value represents a measurement, while a `Z` value usually represents altitude (but can be something like depth in a water based location).

Details

There is no `numeric` input option for multipolygon. There is no way as of yet to make a nested multipolygon with `data.frame` input, but you can do so with list input. See examples.

See Also

Other R-objects: `circularstring()`, `geometrycollection()`, `linestring()`, `multilinestring()`, `multipoint()`, `point()`, `polygon()`

Examples

```
## empty multipolygon
multipolygon("empty")
# multipolygon("stuff")

# data.frame
df <- data.frame(long = c(30, 45, 10, 30), lat = c(20, 40, 40, 20))
df2 <- data.frame(long = c(15, 40, 10, 5, 15), lat = c(5, 10, 20, 10, 5))
multipolygon(df, df2, fmt=0)
lint(multipolygon(df, df2, fmt=0))
wktview(multipolygon(df, df2), zoom=3)

# matrix
mat <- matrix(c(df$long, df$lat), ncol = 2)
mat2 <- matrix(c(df2$long, df2$lat), ncol = 2)
multipolygon(mat, mat2, fmt=0)

# list
multipolygon(list(c(30, 20), c(45, 40), c(10, 40), c(30, 20)),
  list(c(15, 5), c(40, 10), c(10, 20), c(5, 10), c(15, 5)), fmt=2)

polys <- list(
  list(c(30, 20), c(45, 40), c(10, 40), c(30, 20)),
  list(c(15, 5), c(40, 10), c(10, 20), c(5, 10), c(15, 5))
)
wktview(multipolygon(polys, fmt=2), zoom=3)

## nested polygons
polys <- list(
  list(c(40, 40), c(20, 45), c(45, 30), c(40, 40)),
  list(
    list(c(20, 35), c(10, 30), c(10, 10), c(30, 5), c(45, 20), c(20, 35)),
    list(c(30, 20), c(20, 15), c(20, 25), c(30, 20))
  )
)
multipolygon(polys, fmt=0)
lint(multipolygon(polys, fmt=0))



# 3D
## data.frame
df <- data.frame(long = c(30, 45, 10, 30), lat = c(20, 40, 40, 20),
  altitude = 1:4)
df2 <- data.frame(long = c(15, 40, 10, 5, 15), lat = c(5, 10, 20, 10, 5),
  altitude = 1:5)
multipolygon(df, df2, fmt=0)
multipolygon(df, df2, fmt=0, third = "m")
## matrix
mat <- matrix(unname(unlist(df)), ncol = 3)
mat2 <- matrix(unname(unlist(df2)), ncol = 3)
multipolygon(mat, mat2, fmt=0)
multipolygon(mat, mat2, fmt=0, third = "m")
## list
l1 <- list(c(30, 20, 2), c(45, 40, 2), c(10, 40, 2), c(30, 20, 2))
l2 <- list(c(15, 5, 5), c(40, 10, 5), c(10, 20, 5), c(5, 10, 5),
  c(15, 5, 5))
multipolygon(l1, l2, fmt=2)
multipolygon(l1, l2, fmt=2, third = "m")


# 4D
## data.frame
df <- data.frame(long = c(30, 45, 10, 30), lat = c(20, 40, 40, 20),
  altitude = 1:4, weigjht = 20:23)
df2 <- data.frame(long = c(15, 40, 10, 5, 15), lat = c(5, 10, 20, 10, 5),
  altitude = 1:5, weigjht = 200:204)
multipolygon(df, df2, fmt=0)
## matrix
mat <- matrix(unname(unlist(df)), ncol = 4)
mat2 <- matrix(unname(unlist(df2)), ncol = 4)
multipolygon(mat, mat2, fmt=0)
## list
l1 <- list(c(30, 20, 2, 1), c(45, 40, 2, 1), c(10, 40, 2, 1), c(30, 20, 2, 1))
l2 <- list(c(15, 5, 5, 1), c(40, 10, 5, 1), c(10, 20, 5, 1), c(5, 10, 5, 1),
  c(15, 5, 5, 1))
multipolygon(l1, l2, fmt=2)
```

`point` *Make WKT point objects*

Description

Make WKT point objects

Usage

```
point(..., fmt = 16, third = "z")
```

Arguments

`...`	A GeoJSON-like object representing a Point, LineString, Polygon, MultiPolygon, etc.
`fmt`	Format string which indicates the number of digits to display after the decimal point when formatting coordinates. Max: 20

`third`	(character) Only applicable when there are three dimensions. If `m`, assign a `M` value for a measurement, and if `z` assign a `Z` value for three-dimenionsal system. Case is ignored. An `M` value represents a measurement, while a `Z` value usually represents altitude (but can be something like depth in a water based location).

Details

The `third` parameter is used only when there are sets of three points, and you can toggle whether the object gets a `Z` or `M`.

When four points are included, the object automatically gets assigned `ZM`

See Also

Other R-objects: `circularstring()`, `geometrycollection()`, `linestring()`, `multilinestring()`, `multipoint()`, `multipolygon()`, `polygon()`

Examples

```
## empty point
point("empty")
# point("stuff")

## single point
point(-116.4, 45.2)
point(0, 1)

## single point, from data.frame
df <- data.frame(lon=-116.4, lat=45.2)
point(df)

## many points, from a data.frame
ussmall <- us_cities[1:5, ]
df <- data.frame(long = ussmall$long, lat = ussmall$lat)
point(df)

## many points, from a matrix
mat <- matrix(c(df$long, df$lat), ncol = 2)
point(mat)

## single point, from a list
point(list(c(100.0, 3.101)))

## many points, from a list
point(list(c(100.0, 3.101), c(101.0, 2.1), c(3.14, 2.18)))

## when a 3rd point is included
point(1:3, third = "m")
point(1:3, third = "z")
point(list(1:3, 4:6), third = "m")
point(list(1:3, 4:6), third = "z")
point(matrix(1:9, ncol = 3), third = "m")
point(matrix(1:9, ncol = 3), third = "z")
point(data.frame(1, 2, 3), third = "m")
point(data.frame(1, 2, 3), third = "z")
point(data.frame(1:3, 4:6, 7:9), third = "m")

## when a 4th point is included
point(1:4)
point(list(1:4, 5:8))
point(matrix(1:12, ncol = 4))
point(data.frame(1, 2, 3, 4))
point(data.frame(1:3, 4:6, 7:9, 10:12))
```

`polygon` *Make WKT polygon objects*

Description

Make WKT polygon objects

Usage

```
polygon(..., fmt = 16, third = "z")
```

Arguments

`...`	A GeoJSON-like object representing a Point, LineString, Polygon, MultiPolygon, etc.
`fmt`	Format string which indicates the number of digits to display after the decimal point when formatting coordinates. Max: 20
`third`	(character) Only applicable when there are three dimensions. If `m`, assign a `M` value for a measurement, and if `z` assign a `Z` value for three-dimenionsal system. Case is ignored. An `M` value represents a measurement, while a `Z` value usually represents altitude (but can be something like depth in a water based location).

Details

You can create nested polygons with `list` and `data.frame` inputs, but not from `numeric` inputs. See examples.

See Also

Other R-objects: `circularstring()`, `geometrycollection()`, `linestring()`, `multilinestring()`, `multipoint()`, `multipolygon()`, `point()`

Examples

```
## empty polygon
polygon("empty")
# polygon("stuff")

# numeric
polygon(c(100.001, 0.001), c(101.12345, 0.001), c(101.001, 1.001),
  c(100.001, 0.001), fmt=2)

# data.frame
## single polygon
df <- us_cities[2:5,c('long','lat')]
df <- rbind(df, df[1,])
wktview(polygon(df, fmt=2), zoom=4)
## nested polygons
df2 <- data.frame(long = c(-85.9, -85.9, -93, -93, -85.9),
                  lat = c(37.5, 35.3, 35.3, 37.5, 37.5))
wktview(polygon(df, df2, fmt=2), zoom=4)

# matrix
mat <- matrix(c(df$long, df$lat), ncol = 2)
polygon(mat)

# list
# single list - creates single polygon
ply <- list(c(100.001, 0.001), c(101.12345, 0.001), c(101.001, 1.001),
  c(100.001, 0.001))
wktview(polygon(ply, fmt=2), zoom=7)
# nested list - creates nested polygon
vv <- polygon(list(c(35, 10), c(45, 45), c(15, 40), c(10, 20), c(35, 10)),
   list(c(20, 30), c(35, 35), c(30, 20), c(20, 30)), fmt=2)
wktview(vv, zoom=3)
# multiple lists nested within a list
zz <- polygon(list(list(c(35, 10), c(45, 45), c(15, 40), c(10, 20), c(35, 10)),
   list(c(20, 30), c(35, 35), c(30, 20), c(20, 30))), fmt=2)
wktview(zz, zoom=3)


## a 3rd point is included
### numeric
polygon(c(100, 0, 30), c(101, 0, 30), c(101, 1, 30),
  c(100, 0, 30), fmt = 2)
polygon(c(100, 0, 30), c(101, 0, 30), c(101, 1, 30),
  c(100, 0, 30), fmt = 2, third = "m")

### data.frame
df <- us_cities[2:5, c('long','lat')]
df <- rbind(df, df[1,])
df$altitude <- round(runif(NROW(df), 10, 50))
polygon(df, fmt=2)
polygon(df, fmt=2, third = "m")
```

```
### matrix
mat <- matrix(c(df$long, df$lat, df$altitude), ncol = 3)
polygon(mat, fmt=2)
polygon(mat, fmt=2, third = "m")

### list
ply <- list(c(100, 0, 1), c(101, 0, 1), c(101, 1, 1),
  c(100, 0, 1))
polygon(ply, fmt=2)
polygon(ply, fmt=2, third = "m")


## a 4th point is included
### numeric
polygon(c(100, 0, 30, 3.5), c(101, 0, 30, 3.5), c(101, 1, 30, 3.5),
  c(100, 0, 30, 3.5), fmt = 2)

### data.frame
df <- us_cities[2:5, c('long','lat')]
df <- rbind(df, df[1,])
df$altitude <- round(runif(NROW(df), 10, 50))
df$weight <- round(runif(NROW(df), 0, 1), 1)
polygon(df, fmt=2)

### matrix
mat <- matrix(unname(unlist(df)), ncol = 4)
polygon(mat, fmt=2)

### list
ply <- list(c(100, 0, 1, 40), c(101, 0, 1, 44), c(101, 1, 1, 45),
  c(100, 0, 1, 49))
polygon(ply, fmt=2)
```

`properties` *Add properties to a GeoJSON object*

Description

Add properties to a GeoJSON object

Usage

```
properties(x, style = NULL, popup = NULL)
```

Arguments

`x`	(list) GeoJSON as a list
`style`	(list) named list of color, fillColor, etc. attributes. Default: `NULL`
`popup`	(list) named list of popup values. Default: `NULL`

Examples

```
str <- "POINT (-116.4000000000000057 45.2000000000000028)"
x <- wkt2geojson(str)
properties(x, style = list(color = "red"))
```

`us_cities` *This is the same data set from the maps library, named differently*

Description

This database is of us cities of population greater than about 40,000. Also included are state capitals of any population size.

Format

A list with 6 components, namely "name", "country.etc", "pop", "lat", "long", and "capital", containing the city name, the state abbreviation, approximate population (as at January 2006), latitude, longitude and capital status indication (0 for non-capital, 1 for capital, 2 for state capital.

`wkb` *Convert WKT to WKB*

Description

Convert WKT to WKB

Usage

```
wkt_wkb(x)

wkb_wkt(x)
```

Arguments

x	A `character` string representing a WKT object, or an object of class `raw`, representing a WKB object

Value

`wkt_wkb` returns an object of class `raw`, a WKB reprsentation. `wkb_wkt` returns an object of class `character`, a WKT representation

Examples

```
# WKT to WKB
## point
wkt_wkb("POINT (-116.4 45.2)")

## linestring
wkt_wkb("LINESTRING (-116.4 45.2, -118.0 47.0)")

## multipoint
### only accepts the below format, not e.g., ((1 2), (3 4))
wkt_wkb("MULTIPOINT (100.000 3.101, 101.00 2.10, 3.14 2.18)")

## polygon
wkt_wkb("POLYGON ((100.0 0.0, 101.1 0.0, 101.0 1.0, 100.0 0.0))")

# WKB to WKT
## point
(x <- wkt_wkb("POINT (-116.4 45.2)"))
wkb_wkt(x)

## linestring
(x <- wkt_wkb("LINESTRING (-116.4 45.2, -118.0 47.0)"))
wkb_wkt(x)

## multipoint
(x <- wkt_wkb("MULTIPOINT (100.000 3.101, 101.00 2.10, 3.14 2.18)"))
wkb_wkt(x)

## polygon
(x <- wkt_wkb("POLYGON ((100.0 0.0, 101.1 0.0, 101.0 1.0, 100.0 0.0))"))
wkb_wkt(x)
```

`wkt2geojson` *Convert WKT to GeoJSON-like objects.*

Description

Convert WKT to GeoJSON-like objects.

Usage

```
wkt2geojson(str, fmt = 16, feature = TRUE, numeric = TRUE, simplify = FALSE)
```

Arguments

`str`	A GeoJSON-like object representing a Point, LineString, Polygon, MultiPolygon, etc.
`fmt`	Format string which indicates the number of digits to display after the decimal point when formatting coordinates. Max: 20

`feature`	(logical) Make a feature geojson object. Default: `TRUE`
`numeric`	(logical) Give back values as numeric. Default: `TRUE`
`simplify`	(logical) Attempt to simplify from a multi- geometry type to a single type. Applies to multi features only. Default: `FALSE`

Details

Should be robust against a variety of typing errors, including extra spaces between coordinates, no space between WKT type and coordinates. However, some things won't pass, including lowercase WKT types, no spaces between coordinates.

WKT with a 3rd value and when Z is found will be left as is and assumed to be a altitude or similar value. WKT with a 3rd value and when M is found will be discarded as the GeoJSON spec says to do so. WKT with a 4th value as (presumably as a measurement) will also be discarded.

References

https://tools.ietf.org/html/rfc7946

See Also

`geojson2wkt()`

Examples

```
# point
str <- "POINT (-116.4000000000000057 45.2000000000000028)"
wkt2geojson(str)
wkt2geojson(str, feature=FALSE)
wkt2geojson(str, numeric=FALSE)
wkt2geojson("POINT (-116 45)")
wkt2geojson("POINT (-116 45 0)")
## 3D
wkt2geojson("POINT Z(100 3 35)")
wkt2geojson("POINT M(100 3 35)") # dropped if M
## 4D
wkt2geojson("POINT ZM(100 3 35 1.5)") # Z retained

# multipoint
str <- 'MULTIPOINT ((100.000 3.101), (101.000 2.100), (3.140 2.180))'
wkt2geojson(str, fmt = 2)
wkt2geojson(str, fmt = 2, feature=FALSE)
wkt2geojson(str, numeric=FALSE)
wkt2geojson("MULTIPOINT ((100 3), (101 2), (3 2))")
wkt2geojson("MULTIPOINT ((100 3 0), (101 2 0), (3 2 0))")
wkt2geojson("MULTIPOINT ((100 3 0 1), (101 2 0 1), (3 2 0 1))")
## 3D
wkt2geojson("MULTIPOINT Z((100 3 35), (101 2 45), (3 2 89))")
wkt2geojson("MULTIPOINT M((100 3 1.3), (101 2 1.4), (3 2 1.9))")
## 4D
wkt2geojson("MULTIPOINT ZM((100 3 35 0), (101 2 45 0), (3 2 89 0))")
```

```
## simplify
wkt2geojson("MULTIPOINT ((100 3))", simplify = FALSE)
wkt2geojson("MULTIPOINT ((100 3))", simplify = TRUE)


# polygon
str <- "POLYGON ((100 0.1, 101.1 0.3, 101 0.5, 100 0.1),
   (103.2 0.2, 104.8 0.2, 100.8 0.8, 103.2 0.2))"
wkt2geojson(str)
wkt2geojson(str, feature=FALSE)
wkt2geojson(str, numeric=FALSE)
## 3D
str <- "POLYGON Z((100 0.1 3, 101.1 0.3 1, 101 0.5 5, 100 0.1 8),
   (103.2 0.2 3, 104.8 0.2 4, 100.8 0.8 5, 103.2 0.2 9))"
wkt2geojson(str)
## 4D
str <- "POLYGON ZM((100 0.1 3 0, 101.1 0.3 1 0, 101 0.5 5 0, 100 0.1 8 0),
   (103.2 0.2 3 0, 104.8 0.2 4 0, 100.8 0.8 5 0, 103.2 0.2 9 0))"
wkt2geojson(str)


# multipolygon
str <- "MULTIPOLYGON (((40 40, 20 45, 45 30, 40 40)),
 ((20 35, 45 20, 30 5, 10 10, 10 30, 20 35), (30 20, 20 25, 20 15, 30 20)))"
wkt2geojson(str)
wkt2geojson(str, feature=FALSE)
wkt2geojson(str, numeric=FALSE)
## 3D
str <- "MULTIPOLYGON Z(((40 40 1, 20 45 3, 45 30 10, 40 40 0)),
 ((20 35 5, 45 20 67, 30 5 890, 10 10 2, 10 30 0, 20 35 4),
 (30 20 4, 20 25 54, 20 15 56, 30 20 89)))"
wkt2geojson(str)
## 4D
str <- "MULTIPOLYGON ZM(((40 40 1 0, 20 45 3 4, 45 30 10 45, 40 40 0 1)),
 ((20 35 5 8, 45 20 67 9, 30 5 890 89, 10 10 2 234, 10 30 0 5, 20 35 4 1),
 (30 20 4 0, 20 25 54 5, 20 15 56 55, 30 20 89 78)))"
wkt2geojson(str)

# simplify multipolygon to polygon if possible
str <- "MULTIPOLYGON (((40 40, 20 45, 45 30, 40 40)))"
wkt2geojson(str, simplify = FALSE)
wkt2geojson(str, simplify = TRUE)


# linestring
str <- "LINESTRING (100.000 0.000, 101.000 1.000)"
wkt2geojson(str)
wkt2geojson(str, feature = FALSE)
wkt2geojson("LINESTRING (0 -1, -2 -3, -4 5)")
wkt2geojson("LINESTRING (0 1 2, 4 5 6)")
wkt2geojson(str, numeric = FALSE)
## 3D
wkt2geojson("LINESTRING Z(100.000 0.000 3, 101.000 1.000 5)")
```

```
wkt2geojson("LINESTRING M(100.000 0.000 10, 101.000 1.000 67)")
## 4D
wkt2geojson("LINESTRING ZM(100 0 1 4, 101 1 5 78)")


# multilinestring
str <- "MULTILINESTRING ((30 1, 40 30, 50 20)(10 0, 20 1))"
wkt2geojson(str)
wkt2geojson(str, numeric=FALSE)

str <- "MULTILINESTRING (
   (-105.0 39.5, -105.0 39.5, -105.0 39.5, -105.0 39.5,
     -105.0 39.5, -105.0 39.5),
   (-105.0 39.5, -105.0 39.5, -105.0 39.5),
   (-105.0 39.5, -105.0 39.5, -105.0 39.5, -105.0 39.5, -105.0 39.5),
   (-105.0 39.5, -105.0 39.5, -105.0 39.5, -105.0 39.5))"
wkt2geojson(str)
wkt2geojson(str, numeric=FALSE)

## 3D
str <- "MULTILINESTRING Z((30 1 0, 40 30 0, 50 20 0)(10 0 1, 20 1 1))"
wkt2geojson(str)
str <- "MULTILINESTRING M((30 1 0, 40 30 0, 50 20 0)(10 0 1, 20 1 1))"
wkt2geojson(str)
## 4D
str <- "MULTILINESTRING ZM((30 1 0 5, 40 30 0 7, 50 20 0 1)(10 0 1 1, 20 1 1 1))"
wkt2geojson(str)

# simplify multilinestring to linestring if possible
str <- "MULTILINESTRING ((30 1, 40 30, 50 20))"
wkt2geojson(str, simplify = FALSE)
wkt2geojson(str, simplify = TRUE)


# Geometrycollection
str <- "GEOMETRYCOLLECTION (
 POINT Z(0 1 4),
 LINESTRING (-100 0, -101 -1),
 POLYGON ((100.001 0.001, 101.1235 0.0010, 101.001 1.001, 100.001 0.001),
           (100.201 0.201, 100.801 0.201, 100.801 0.801, 100.201 0.201)),
 MULTIPOINT ((100.000 3.101), (101.0 2.1), (3.14 2.18)),
 MULTILINESTRING ((0 -1, -2 -3, -4 -5),
       (1.66 -31.50, 10.0 3.0, 10.9 1.1, 0.0 4.4)),
 MULTIPOLYGON (((100.001 0.001, 101.001 0.001, 101.001 1.001, 100.001 0.001),
               (100.201 0.201, 100.801 0.201, 100.801 0.801, 100.201 0.201)),
                 ((1 2 3, 5 6 7, 9 10 11, 1 2 3))))"
wkt2geojson(str)
wkt2geojson(str, numeric=FALSE)

# case doesn't matter
str <- "point (-116.4000000000000057 45.2000000000000028)"
wkt2geojson(str)
```

wktview *Visualize geojson from a character string or list*

Description

Visualize geojson from a character string or list

Usage

```
wktview(x, center = NULL, zoom = 5, fmt = 16)
```

Arguments

`x`	Input, a geojson character string or list
`center`	(numeric) A length two vector of the form: longitude, latitude
`zoom`	(integer) A number between 1 and 18 (1 zoomed out, 18 zoomed in)
`fmt`	Format string which indicates the number of digits to display after the decimal point when formatting coordinates. Max: 20

Value

Opens a map with the geojson object(s) using `leaflet`

See Also

`as_featurecollection()`

Examples

```
## Not run:
# point
str <- "POINT (-116.4000000000000057 45.2000000000000028)"
wktview(str)

# multipoint
df <- us_cities[1:5,c('long','lat')]
str <- multipoint(df)
wktview(str, center = c(-100,40))
wktview(str, center = c(-100,40), zoom = 3)

# linestring
wktview(linestring(c(100.000, 0.000), c(101.000, 1.000), fmt=2),
  center = c(100, 0))

# polygon
a <- polygon(list(c(100.001, 0.001), c(101.12345, 0.001), c(101.001, 1.001),
  c(100.001, 0.001)))
wktview(a, center = c(100, 0))
wktview(a, center = c(100.5, 0), zoom=9)

## End(Not run)
```

Index