Package ‘mregions’

October 17, 2017

Title Marine Regions Data from 'Marineregions.org'

Description Tools to get marine regions data from
<http://www.marineregions.org/>. Includes tools to get region metadata,
as well as data in 'GeoJSON' format, as well as Shape files. Use cases
include using data downstream to visualize 'geospatial' data by marine
region, mapping variation among different regions, and more.

Version 0.1.6

License MIT + file LICENSE

URL https://github.com/ropenscilabs/mregions

BugReports https://github.com/ropenscilabs/mregions/issues

Imports httr (>= 1.1.0), jsonlite (>= 1.0), xml2, wellknown, rappdirs,
sp, data.table, tibble

Suggests roxygen2 (>= 6.0.1), testthat, geojsonio, rgdal, rgeos,
knitr, rmapshaper

Enhances leaflet

VignetteBuilder knitr

RoxygenNote 6.0.1

NeedsCompilation no

Author Scott Chamberlain [aut, cre],
Francois Michonneau [ctb],
Pieter Provoost [ctb],
Michael Sumner [ctb]

Maintainer Scott Chamberlain <myrmecocystus@gmail.com>

Repository CRAN

Date/Publication 2017-10-17 21:09:26 UTC

R topics documented:

mregions-package .......................................................... 2
mr_as_wkt ................................................................. 3
mregions-package

描述

海洋区域数据来自Marineregions

详情

mregions从<http://www.marineregions.org/>获取数据

用例

mregions对多样化的R用户有用，因为您可以访问MarineRegions拥有的所有数据，这可以在多种用例中提供帮助:

- 单独可视化海洋区域
- 结合相关数据和分析可视化海洋区域
- 使用海洋区域的空间边界查询数据提供商（例如OBIS<http://www.iobis.org/>）
- 通过地名获取地理定位数据
- 通过地理定位数据获取地名

作者

Scott Chamberlain <myrmecocystus@gmail.com>
**mr_as_wkt**

## Examples

```r
## Not run:
## GeoJSON
## Get region
res <- mr_geojson(key = "Morocco:dam")

## Plot data
if (!requireNamespace("leaflet")) {
  install.packages("leaflet")
}
library("leaflet")
leaflet()
  addProviderTiles(provider = 'OpenStreetMap')
  addGeoJSON(geojson = res$features)
  setView(-3.98, 35.1, zoom = 11)

## Shape
## Get region
res <- mr_shp(key = "MarineRegions:eez_iho_union_v2")
library("leaflet")
leaflet()
  addProviderTiles(provider = 'OpenStreetMap')
  addPolygons(data = res)

## Convert to WKT
## From geojson
res <- mr_geojson(key = "Morocco:dam")
mr_as_wkt(res, fmt = 5)

## From shp object ('SpatialPolygonsDataFrame') or file, both work
mr_as_wkt(mr_shp(key = "Morocco:dam", read = FALSE))

## End(Not run)
```

---

### mr_as_wkt

**Convert data to WKT**

#### Description

Convert data to WKT

#### Usage

```r
mr_as_wkt(x, fmt = 16, ...)
```
Arguments

- `x`: Output from `mr_geojson`, `mr_shp`, or a SpatialPolygonsDataFrame
- `fmt` (integer): The number of digits to display after the decimal point when formatting coordinates. Ignored when shp files or SpatialPolygonsDataFrame passed in
- `...`: Further args passed on to `fromJSON` only in the event of json passed as a character string. Ignored when shp files or SpatialPolygonsDataFrame passed in

Details

WKT, or Well Known Text, is a way to encode spatial data. It's somewhat similar to GeoJSON, but instead of being in JSON format, it's a character string (though can also be encoded in binary format). WKT is often used in SQL databases, and many species occurrence APIs allow only WKT. You could do the conversion to WKT yourself, but we provide `as_wkt` as a convenience.

Value

A character string of WKT data

Examples

```r
## Not run:
res <- mr_geojson(key = "Morocco:dam")
mr_as_wkt(res, fmt = 5)

# shp files
## path to wkt
mr_as_wkt(mr_shp(key = "Morocco:dam", read = FALSE))

## spatial object to wkt
mr_as_wkt(mr_shp(key = "Morocco:dam", read = TRUE))

## End(Not run)
```

---

`mr_features_get` | Get features

Description

Get features

Usage

`mr_features_get(type, featureID, maxFeatures = 100, format = "json", path = NULL, version = "2.0.0", ...)"
**Arguments**

- **type** (character): a region type, e.g., "MarineRegions:eez". required
- **featureID** (character): a feature ID. required
- **maxFeatures** (integer): Number of features. Default: 100
- **format** (character): output format, see Details for allowed options. Default: json
- **path** (character): required when format="SHAPE-ZIP", otherwise, ignored
- **version** (character): either 1.0.0 or 2.0.0 (default). In v1.0.0, the coordinates are in format y,x (long, lat), while in 2.0.0 the coordinates are in format x,y (lat, long)

...Curl options passed on to GET

**Details**

Allowed options for the format parameter:

- text/xml; subtype=gml/3.2
- GML2
- KML
- SHAPE-ZIP
- application/gml+xml; version=3.2
- application/json
- application/vnd.google-earth.kml+xml
- application/vnd.google-earth.kml+xml
- csv
- gml3
- gml32
- json
- text/xml; subtype=gml/2.1.2
- text/xml; subtype=gml/3.1.1

**Value**

depends on the format option used, usually a text string

**Examples**

```r
## Not run:
# json by default
mr_features_get(type = "MarineRegions:eez", featureID = "eez.3")
# csv
mr_features_get(type = "MarineRegions:eez", featureID = "eez.3", format = "csv")
# KML
mr_features_get(type = "MarineRegions:eez", featureID = "eez.3", format = "KML")
```
mr_geojson

Get a Marineregions geojson file

Description

Get a Marineregions geojson file

Usage

mr_geojson(key = NULL, name = NULL, maxFeatures = 50, ...)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>key</td>
<td>(character) Region key, of the form <code>x:y</code>, where <code>x</code> is a namespace (e.g., MarineRegions), and <code>y</code> is a region (e.g., eez_33176)</td>
</tr>
<tr>
<td>name</td>
<td>(character) Region name, if you supply this, we search against titles via <code>mr_names</code> function</td>
</tr>
<tr>
<td>maxFeatures</td>
<td>(integer) Number of features to return. Default: 50</td>
</tr>
<tr>
<td>...</td>
<td>Curl options passed on to <code>GET</code></td>
</tr>
</tbody>
</table>

Value

an S3 class of type `mr_geojson`, just a thin wrapper around a list. The list has names:

- type (character) - the geojson type (e.g., FeatureCollection)
- totalFeatures (integer) - the...
• features (list) - the features, with slots for each feature: type, id, geometry, geometry_name, and properties
• crs (list) - the coordinate reference system
• bbox (list) - the bounding box that encapsulates the object

Examples

```r
## Not run:
# by key
res1 <- mr_geojson(key = "Morocco:dam")

# by name -- not working right now
if (requireNamespace("geojsonio")) {
  library("geojsonio")
  as.json(unclass(res1)) %>% map_leaf

  # MEOW - marine ecoregions
  as.json(unclass(mr_geojson("Ecoregions:ecoregions"))) %>% map_leaf()
}
## End(Not run)
```

---

 mr_geo_code

Geocode with Marineregions

Description

Geocode with Marineregions

Usage

mr_geo_code(place, like = TRUE, fuzzy = FALSE, ...)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>place</td>
<td>(character) a place name</td>
</tr>
<tr>
<td>like</td>
<td>(logical) adds a percent-sign before and after place value (a SQL LIKE function). Default: TRUE</td>
</tr>
<tr>
<td>fuzzy</td>
<td>(logical) Uses Levenshtein query to find nearest matches. Default: FALSE</td>
</tr>
<tr>
<td>...</td>
<td>Curl options passed on to GET</td>
</tr>
</tbody>
</table>
Value

If no results, an empty list. If results found, a data.frame with the columns:

- MRGID (integer)
- gazetteerSource (character)
- placeType (character)
- latitude (numeric)
- longitude (numeric)
- minLatitude (numeric)
- minLongitude (numeric)
- maxLatitude (numeric)
- maxLongitude (numeric)
- precision (numeric)
- preferredGazetteerName (character)
- preferredGazetteerNameLang (character)
- status (character)
- accepted (integer)

Examples

```r
## Not run:
# search for 'oost', like=TRUE, and not fuzzy
mr_geo_code(place = "oost", like = TRUE, fuzzy = FALSE)

# search for 'oost', like=FALSE, and not fuzzy
mr_geo_code(place = "oost", like = FALSE, fuzzy = FALSE)

# search for 'oost', like=FALSE, and fuzzy
mr_geo_code(place = "oost", like = FALSE, fuzzy = TRUE)

# search for 'oost', like=TRUE, and fuzzy
mr_geo_code(place = "oost", like = TRUE, fuzzy = TRUE)

# search for 'ast', like=TRUE, and fuzzy
mr_geo_code(place = "ast", like = TRUE, fuzzy = TRUE)

## End(Not run)
```
**mr_layers**

*Description*

list layers

*Usage*

mr_layers(...)

*Arguments*

... Curl options passed on to `GET`

*Examples*

```r
## Not run:
res <- mr_layers()
vapply(res, ['[', '', 'Name'])
## End(Not run)
```

---

**mr_names**

*Description*

Get region names - v2

*Usage*

mr_names(layer, ...)

*Arguments*

layer A layer name, one of MarineRegions:eez, MarineRegions:eez_boundaries, MarineRegions:iho, MarineRegions:fao, or MarineRegions:lme

... Curl options passed on to `GET`
mr_names_search

Search for region names

Description
Search for region names

Usage
mr_names_search(x, q = NULL, ...)

Arguments

x, q
Either a tbl_df, returned from mr_names, or a query as a character string. If a
tbl_df, you must pass a query string to q. If a query string (character) is passed
to x, leave q as NULL

... Parameters passed on to agrep

Value
NULL if no matches found, or a data.frame, or tibble, of class tbl_df, with slots:

- name (character) - name of the region, which is a combination of the name_first and name_second,
e.g., Morocco:elevation_10m
- title (character) - title for the region
- name_first (character) - first part of the name, e.g., Morocco
- name_second (character) - second part of the name, e.g., elevation_10m
mr_obis_eez_id

## Examples

```r
## Not run:
# Get region names with mr_names() function
(res <- mr_names("MarineRegions:eez"))

# to save time, pass in the result from mr_names()
mr_names_search(res, q = "Amer")

# if you don't pass in the result from mr_names(), we have to
# call mr_names() internally, adding some time
mr_names_search(x = "iho", q = "Black")
mr_names_search(x = "iho", q = "Sea")

# more examples
mr_names_search("iho", "Sea")
(res <- mr_names("MarineRegions:iho"))
mr_names_search(res, q = "Sea")

## End(Not run)
```

---

### Description

Get OBIS EEZ id

### Usage

```r
mr_obis_eez_id(x)
```

### Arguments

- `x` (character) An Exclusive Economic Zone name

### Details

Internally we use the OBIS API to retrieve an EEZ id.

Matching internally is case insensitive, as we convert your input and match against EEZ names that are all lower case.

### Value

An integer EEZ ID if a match found in list of EEZ’s, or NULL if no match found.
Examples

```r
## Not run:
# You can get EEZ names via the mr_names() function
(res <- mr_names('MarineRegions:eez_boundaries'))
mr_obis_eez_id(res$eez1[19])

# Or pass in a name
mr_obis_eez_id("Bulgarian Exclusive Economic Zone")

# case doesn't matter
mr_obis_eez_id("bulgarian exclusive economic zone")

# No match, gives NULL
mr_obis_eez_id("stuff things")

## End(Not run)
```

### mr_place_relations

**Related records**

Get related records based on their MRGID.

#### Usage

```r
mr_place_relations(mrgid, direction = c("upper", "lower", "both"),
                  type = c("partof", "partlypartof", "adjacentto", "similarto",
                           "administrativepartof", "influencedby", "all"), ...)
```

#### Arguments

- **mrgid** (numeric) the MRGID (Marineregions Global Identifier) for the record of interest
- **direction** (character) in which direction of the geographical hierarchy should the records be retrieved? Default: upper
- **type** (character) what kind of relations should the records retrieve have with the place? Default: partof
- ... curl options to be passed on to GET

#### Author(s)

Francois Michonneau <francois.michonneau@gmail.com>
Examples

## Not run:
## geocode to get geospatial data for a place name
(tikehau <- mr_geo_code("tikehau"))

## then pass in an MRGID as the first parameter
mr_place_relations(tikehau$MRGID)

## Set direction='both'
mr_place_relations(tikehau$MRGID, direction = "both")

## Set type to various other options
mr_place_relations(307, type = "adjacentto")
mr_place_relations(414, type = "similarto")
mr_place_relations(4177, type = "all")

## End(Not run)

---

**mr_place_types**  
*Get Marineregions place types*

### Description

Get Marineregions place types

### Usage

mr_place_types(...)

### Arguments

...  
Curl options passed on to GET

### Value

A data.frame with the columns:

- type (character) the place type
- description (character) description of the place type

### Examples

## Not run:
res <- mr_place_types()
head(res)
res$type

## End(Not run)
**mr_records_by_type**

Get Marineregions records by place type

**Description**

Get Marineregions records by place type

**Usage**

`mr_records_by_type(type, offset = 0, ...)`

**Arguments**

- **type** (character) One place type name. See `mr_place_types` for place type names
- **offset** (numeric) Offset to start at. Each request can return up to 100 results. e.g., an offset of 200 will give records 200 to 299.
- **...** Curl options passed on to `GET`

**Details**

Internally we use the `getGazetteerRecordsByType`.json API method, which searches for Marineregions records by user supplied place type

**Value**

If no results, an empty list. If results found, a data.frame with the columns:

- MRGID (integer)
- gazetteerSource (character)
- placeType (character)
- latitude (numeric)
- longitude (numeric)
- minLatitude (numeric)
- minLongitude (numeric)
- maxLatitude (numeric)
- maxLongitude (numeric)
- precision (numeric)
- preferredGazetteerName (character)
- preferredGazetteerNameLang (character)
- status (character)
- accepted (integer)
mr_rev_geo_code

Examples

```r
## Not run:
# Get records of type 'EEZ', then inspect data.frame
res <- mr_records_by_type(type="EEZ")
head(res)

# You can use mr_place_types() function to get types
# then pass those into this function
types <- mr_place_types()
mr_records_by_type(types$type[1])
mr_records_by_type(types$type[10])

# use regex to find a type name matching a pattern
x <- grep("MEOW", types$type, value = TRUE)

# then pass to the function
mr_records_by_type(x)
mr_records_by_type(x, offset = 100)

## End(Not run)
```

---

**Description**

Retrieve the names of geographic objects from coordinates (and optionally a radius around them).

**Usage**

```
mr_rev_geo_code(lat, lon, lat_radius = 1, lon_radius = 1, ...)
```

**Arguments**

- **lat** (numeric) Latitude for the coordinates (decimal format)
- **lon** (numeric) Longitude for the coordinates (decimal format)
- **lat_radius** (numeric) Extends search to include the range from lat-lat_radius to lat+lat_radius
- **lon_radius** (numeric) Extends search to include the range from lon-lon_radius to lon+lon_radius
- ... curl options to be passed on to GET

**Value**

If no results, an empty list. If results found, a data.frame with the columns:

- MRGID (integer)
- gazetteerSource (character)
- placeType (character)
• latitude (numeric)
• longitude (numeric)
• minLatitude (numeric)
• minLongitude (numeric)
• maxLatitude (numeric)
• maxLongitude (numeric)
• precision (numeric)
• preferredGazetteerName (character)
• preferredGazetteerNameLang (character)
• status (character)
• accepted (integer)

Author(s)
Francois Michonneau <francois.michonneau@gmail.com>

Examples

```r
## Not run:
# Setting radius to 0.5
mr_rev_geo_code(-21.5, 55.5, lat_radius=0.5, lon_radius=0.5)

# radius to 3
mr_rev_geo_code(-21.5, 55.5, lat_radius=3, lon_radius=3)

# radius to 1
mr_rev_geo_code(-15, 45, lat_radius=1, lon_radius=1)

## End(Not run)
```

---

`mr_shp`  
*Get a region shp file*

**Description**

Get a region shp file

**Usage**

```r
mr_shp(key = NULL, name = NULL, maxFeatures = 50, overwrite = TRUE,  
read = TRUE, filter = NULL, ...)
```
Arguments

- **key** (character) Region key, of the form x:y, where x is a namespace (e.g., MarineRegions), and y is a region (e.g., eez_33176)
- **name** (character) Region name, if you supply this, we search against titles via `mr_names` function
- **maxFeatures** (integer) Number of features
- **overwrite** (logical) Overwrite file if already exists. Default: FALSE
- **read** (logical) To read in as spatial object. If FALSE a path given back. if TRUE, you need the rgdal package installed. Default: FALSE
- **filter** (character) String to filter features on

... Curl options passed on to `GET`. since we use caching, note that if you've made the exact same request before and the file is still in cache, we grab the cached file and don’t make an HTTP request, so any curl options passed would be ignored.

Details

We use `rappdirs` to determine where to cache data depending on your operating system. See `rappdirs::user_cache_dir("mregions")` for location on your machine

We cache based on the name of the region plus the `maxFeatures` parameter. That is to say, you can query the same region name, but with different `maxFeatures` parameter values, and they will get cached separately. You can clear the cache by going to the directory at `rappdirs::user_cache_dir("mregions")` and deleting the files.

We use `stringsAsFactors = FALSE` inside of `rgdal::readOGR()` so that character variables aren't converted to factors.

Value

A `SpatialPolygonsDataFrame` if `read = TRUE`, or a path to a SHP file on disk if `read = FALSE`.

Note

the parameter `name` is temporarily not useable. MarineRegions updated their web services, and we haven't sorted out yet how to make this feature work. We may bring it back in future version of this package.

Examples

```r
## Not run:
## just get path
mr_shp(key = "MarineRegions:eez_iho_union_v2", read = FALSE)
## read shp file into spatial object
res <- mr_shp(key = "MarineRegions:eez_iho_union_v2", read = TRUE)

mr_shp(key = "SAIL:w_marinehabitatd")

# maxFeatures
library(sp)
```
plot(mr_shp(key = "MarineRegions:eez_iho_union_v2"))
plot(mr_shp(key = "MarineRegions:eez_iho_union_v2", maxFeatures = 5))

# visualize with package leaflet
if (requireNamespace("leaflet")) {
  library("leaflet")
  leaflet() %>%
    addTiles() %>%
    addPolygons(data = res)
}

# use `filter` param to get a subset of a region
library(sp)
pp <- mr_shp(key = "MarineRegions:eez_iho_union_v2")
plot(pp)
rr <- mr_shp(key = "MarineRegions:eez_iho_union_v2",
             filter = "North Atlantic Ocean")
plot(rr)

# get Samoan Exclusive Economic Zone
res <- mr_shp(
  key = "MarineRegions:eez",
  filter = "Samoan Exclusive Economic Zone"
)
sp::plot(res)

## End(Not run)
Index

*Topic package
  mregions-package, 2

agrep, 10

fromJSON, 4

GET, 5–7, 9, 12–15, 17

mr_as_wkt, 3
mr_features_get, 4
mr_geo_code, 7
mr_geojson, 4, 6
mr_layers, 9
mr_names, 6, 9, 10, 17
mr_names_search, 10
mr_obis_eez_id, 11
mr_place_relations, 12
mr_place_types, 13, 14
mr_records_by_type, 14
mr_rev_geo_code, 15
mr_shp, 4, 16
mregions (mregions-package), 2
mregions-package, 2