Package ‘happign’

October 13, 2022

Title R Interface to ‘IGN’ Web Services
Version 0.1.6
Maintainer Paul Carteron <carteronpaul@gmail.com>
Description Interface to easily access the National Institute of Geographic and Forestry Information open-source data from Geoservice website for any area of interest in France via WFS (shapefile) and WMS (raster) web services
<https://geoservices.ign.fr/services-web-experts>.
License GPL (>= 3)
URL https://github.com/paul-carteron,
https://paul-carteron.github.io/happign/
BugReports https://github.com/paul-carteron/happign/issues
Depends R (>= 3.3.0)
Imports archive, checkmate, curl, dplyr, htr2, magrittr, methods, sf
(>= 1.0-7), terra, xml2
Suggests covr, DT, httptest2, knitr, rmarkdown, testthat (>= 3.0.0),
tmap, stars, purrr
VignetteBuilder knitr
Config/testthat/edition 3
Encoding UTF-8
LazyData true
NeedsCompilation no
RoxygenNote 7.2.1
SystemRequirements C++11, GDAL (>= 2.0.1), GEOS (>= 3.4.0), PROJ (>= 4.8.0), sqlite3
Author Paul Carteron [aut, cre] (<https://orcid.org/0000-0002-6942-6662>)
Repository CRAN
Date/Publication 2022-09-16 11:06:10 UTC
**R topics documented:**

- `are_queryable` .............................. 2
- `cog_2022` ..................................... 3
- `get_apicarto_cadastre` ...................... 3
- `get_apicarto_commune` .................... 5
- `get_apicarto_plu` ......................... 6
- `get_apikeys` ................................. 8
- `get_layers_metadata` ..................... 8
- `get_raw_lidar` ............................... 10
- `get_wfs` ..................................... 11
- `get_wms_info` .............................. 12
- `get_wms_raster` ......................... 13
- `shp_to_geojson` ............................ 15

**Index**

| are_queryable | Check if a wms layer is queryable with GetFeatureInfo |

**Description**

Check if a wms layer is queryable with GetFeatureInfo

**Usage**

`are_queryable(apikey)`

**Arguments**

- `apikey`  
  API key from `get_apikeys()` or directly from the IGN website

**Value**

character containing the name of the queryable layers

**See Also**

`get_wms_info()`
Description

A dataset containing insee code and libelle of commune as of January 1, 2022. COG mean Code Officiel Géographique

Usage

cog_2022

Format

A vector

Source

https://www.insee.fr/fr/information/2115000

Description

Implementation of the cadastre module of the IGN's apicarto

Usage

```r
## S3 method for class 'sf'
get_apicarto_cadastre(
  x,
ssection = NULL,
numero = NULL,
code_abs = NULL,
source_ign = "PCI"
)

## S3 method for class 'sfc'
get_apicarto_cadastre(
  x,
ssection = NULL,
numero = NULL,
code_abs = NULL,
source_ign = "PCI"
)
```
get_apicarto_cadastre

## S3 method for class 'character'
get_apicarto_cadastre(
  x,
  section = NULL,
  numero = NULL,
  code_abs = NULL,
  source_ign = "PCI"
)

Arguments

x  It can be a shape or multiple insee code :
  • Shape : all the cadastral parcels contained in it are downloaded. It should be an object of class sf or sfc.
  • Code insee : filter the response on the INSEE code entered (must be a character or a vector of character)
section  A character or a vector of character to filter the response on the cadastral section code entered (on 2 characters)
numero  A character or a vector of character to filter the answers on the entered parcel number (on 4 characters)
code_abs  A character or a vector of character to filter the answers on the code of absorbed commune. This prefix is useful to differentiate between communes that have merged
source_ign  Can be "BDP" for BD Parcellaire or "PCI" for Parcellaire express. The BD Parcellaire is a discontinued product. Its use is no longer recommended because it is no longer updated. The use of PCI Express is strongly recommended and will become mandatory. More information on the comparison of this two products can be found here

Details

#' @usage get_apicarto_cadastre(x, section = NULL, numero = NULL, code_abs = NULL, source_ign = "PCI")

Value

get_apicarto_cadastre return an object of class sf

Examples

## Not run:
library(sf)
library(tmap)

# line from the best town in France
line <- st_linestring(matrix(c(-4.372215, -4.365177, 47.803943, 47.79772),
    ncol = 2))
get_apicarto_commune

```r
line <- st_sfc(line, crs = st_crs(4326))

PCI_shape <- get_apicarto_cadastre(shape, section = c("AX", "AV"))
BDP_Code <- get_apicarto_cadastre("29158", section = c("AX", "BR"),
  source_ign = "BDP")

tm_shape(PCl_shape)+
  tm_borders()+
  tm_lines(col = "red")

tm_shape(BDP_Code)+
  tm_polygons(col = "section", border.col = "black")

## End(Not run)
```

---

**get_apicarto_commune**  
*Apicarto Commune*

---

**Description**

Implementation of the cadastre module of the IGN’s apicarto

**Usage**

```r
get_apicarto_commune(x,
  source_ign = "PCI")
```

**Arguments**

- `x`  
  It can be a shape, insee code or departement code.
  - shape : it must be an object of class sf or sfc.
  - insee or departement code : it must be an object of class character. All
    insee code from France can be retrieved by running data(cog_2022)

- `source_ign`  
  Can be "BDP" for BD Parcellaire or "PCI" for Parcellaire express. The BD Parcellaire is a discontinued product. Its use is no longer recommended because it is no longer updated. The use of PCI Express is strongly recommended and will become mandatory. More information on the comparison of this two products can be found [here](#)

**Value**

get_apicarto_commune return an object of class sf
get_apicarto_plu

### Examples

```r
## Not run:
library(sf)
library(tmap)

# line from the best town in France
line <- st_linestring(matrix(c(-4.372215, -4.365177, 47.803943, 47.79772),
                            ncol = 2))
line <- st_sfc(line, crs = st_crs(4326))

commune <- get_apicarto_commune(line)

tm_shape(commune)+
  tm_borders()+
  tm_shape(line)+
  tm_lines(col = "red", lwd = 2)

# code_insee of the best town in France
commune <- get_apicarto_commune("29158")

tm_shape(commune)+
  tm_borders()+
  tm_text("nom_com")

## End(Not run)
```

---

get_apicarto_plu  
*Apicarto module Geoportail de l’urbanisme*

---

**Description**

Apicarto module Geoportail de l’urbanisme

**Usage**

```r
get_apicarto_plu(x, 
  ressource = "zone-urba",
  partition = NULL)
```

**Arguments**

- **x**: An object of class `sf` or `sfc`. If NULL, partition must be filled by partition of PLU.
- **partition**: A character corresponding to PLU partition (can be retrieve using `get_apicarto_plu(x, "document", partition = NULL)`). If partition is explicitly set, all PLU features are returned and geom is override.
get_apicarto_plu

Details

- "municipality": information on the communes (commune with RNU, merged commune)
- "document": information on urban planning documents (POS, PLU, PLUi, CC, PSMV)
- "zone-urba": zoning of urban planning documents,
- "secteur-cc": communal map sectors
- "prescription-surf": surface prescriptions like Classified wooded area, Area contributing to the green and blue framework, Landscape element to be protected or created, Protected open space, ...
- "prescription-lin": linear prescription like pedestrian path, bicycle path, hedges or tree lines to be protected, ...
- "prescription-pct": punctual prescription like Building of architectural interest, Building to protect, Remarkable tree, Protected pools, ...
- "info-surf": surface information perimeters of urban planning documents like Protection of drinking water catchments, archaeological sector, noise classification, ...
- "info-lin": linear information perimeters of urban planning documents like Bicycle path to be created, Long hike, Façade and/or roof protected as historical monuments, ...
- "info-pct": punctual information perimeters of urban planning documents like Archaeological heritage, Listed or classified historical monument, Underground cavity, ...

Value

A object of class sf

Examples

```r
## Not run:
library(tmap)
library(sf)
point <- st_sfc(st_point(c(-0.4950188466302029, 45.428039987269926)), crs = 4326)

# If you know the partition (all PLU features are returned, geom is override)
partition <- "DU_17345"
poly <- get_apicarto_plu(x = NULL, ressource = "zone-urba", partition = partition)
qtm(poly)+qtm(point, symbols.col = "red", symbols.size = 2)

# If you don't know partition (only intersection between geom and PLU features is returned)
poly <- get_apicarto_plu(x = point, ressource = "zone-urba", partition = NULL)
qtm(poly)+qtm(point, symbols.col = "red", symbols.size = 2)

# If you wanna find partition
document <- get_apicarto_plu(point, ressource = "document", partition = NULL)
partition <- unique(document$partition)

# Get all prescription : !\ prescription is different than zone-urba
partition <- "DU_17345"
ressources <- c("prescription-surf", "prescription-lin", "prescription-pct")

library(purrr)
```


```r
all_prescription <- map(.x = ressources,
                        .f = ~ get_apicarto_plu(point, .x, partition))
```

## End(Not run)

---

**get_apikeys**

*List of all API keys from IGN*

**Description**

All API keys are manually extracted from this table provided by IGN.

**Usage**

```r
get_apikeys()
```

**Value**

*character*

**Examples**

```r
## Not run:
# One API key
get_apikeys()[1]

# All API keys
get_apikeys()

## End(Not run)
```

---

**get_layers_metadata**

*Metadata for one couple of apikey and data_type*

**Description**

Metadata are retrieved using the IGN APIs. The execution time can be long depending on the size of the metadata associated with the API key and the overload of the IGN servers.

**Usage**

```r
get_layers_metadata(apikey, data_type)
```
get_layers_metadata

Arguments

apikey API key from get_apikeys() or directly from the IGN website

data_type Should be "wfs" or "wms". See details for more information about these two Webservice formats.

Details

• WFS is a standard protocol defined by the OGC (Open Geospatial Consortium) and recognized by an ISO standard. The reference document is available on the OGC website. The Geoportail WFS service implements version 2.0 of this protocol. The WFS service of Geoportail gives access to objects from different IGN databases: BD TOPO®, BD CARTO®, BD ADRESSE®, BD FORET® or PARCELLAIRE EXPRESS (PCI).

• WMS is a standard protocol defined by the OGC (Open Geospatial Consortium) and recognized by an ISO standard. The reference document is available on the OGC website.

• For further more detail, check IGN documentation page

Value
data.frame

See Also
get_apikeys()

Examples

```r
## Not run:
apikey <- get_apikeys()[4]
metadata_table <- get_layers_metadata(apikey, "wms")
all_layer_name <- metadata_table$name
abstract_of_MNT <- metadata_table[1,"abstract"]

# list with every wfs metadata (warning : it's quite long)
list_metadata <- lapply(X = get_apikeys(),
                        FUN = get_layers_metadata,
                        data_type = "wfs")

# Convert list to one single data.frame
all_metadata <- dplyr::bind_rows(list_metadata)

## End(Not run)
```
get_raw_lidar

Description

The raw LIDAR data are not classified. They correspond to a point cloud.

Usage

get_raw_lidar(shape, destfile = ".", grid_path = ".", quiet = F)

Arguments

- **shape**: Object of class `sf`. Needs to be located in France.
- **destfile**: folder path where data are downloaded. By default set to "." e.g. the current directory
- **grid_path**: folder path where grid is downloaded. By default set to "." e.g. the current directory
- **quiet**: if TRUE download is silent

Details

get_raw_lidar first downloads a grid containing the name of LIDAR tiles which is then intersected with shape to determine which ones will be uploaded. The grid is downloaded to grid_path if it is not already in the folder.

get_raw_lidar automatically compares the required tiles to those already in the destfile to avoid re-downloading them.

Value

No object

Examples

```r
## Not run:
library(sf)

# create shape
shape <- st_polygon(list(matrix(c(8.852234, 42.55466,
                   8.852234, 42.57289,
                   8.860474, 42.57289,
                   8.860474, 42.55466,
                   8.852234, 42.55466,
                   ncol = 2, byrow = TRUE)))
shape <- st_sfc(shape, crs = st_crs(4326))

# Download data to current directory
```
get_wfs

get_raw_lidar(shape)

# Check all .laz file
list.files(".", pattern = ".laz", recursive = TRUE)

## End(Not run)

get_wfs

Download WFS layer

Description

Directly download a shapefile layer from the French National Institute of Geographic and Forestry. To do that, it needs a location giving by a shapefile, an apikey and the name of layer. You can find those information from IGN website

Usage

get_wfs(shape,
        apikey,
        layer_name,
        filename = NULL)

Arguments

shape Object of class sf. Needs to be located in France.
apikey API key from get_apikeys() or directly from IGN website
layer_name Name of the layer from get_layers_metadata(apikey, "wfs") or directly from IGN website
filename Either a character string naming a file or a connection open for writing. (ex: "test.shp" or "~/test.shp")

Value

get_wfs return an object of class sf

See Also

get_apikeys(), get_layers_metadata()
Examples

```r
## Not run:
library(sf)
library(tmap)

# Get the borders of best town in France -------------------------
apikey <- get_apikeys()[1]
metadata_table <- get_layers_metadata(apikey, "wfs")
layer_name <- as.character(metadata_table[32,2])

# One point from the best town in France
shape <- st_point(c(-4.373937, 47.79859))
shape <- st_sfc(shape, crs = st_crs(4326))

# Download borders
borders <- get_wfs(shape, apikey, layer_name)

# Verif
tmap_mode("view") # easy interactive map
qtm(borders, fill = NULL, borders = "firebrick") # easy map

# Get forest_area of the best town in France ---------------------
forest_area <- get_wfs(shape = borders,
                       apikey = get_apikeys()[9],
                       layer_name = "LANDCOVER.FORESTINVENTORY.1:resu_bdvi_shape")

# Verif
qtm(forest_area, fill = "libelle")

# Get roads of the best town in France --------------------------
roads <- get_wfs(shape = borders,
                 apikey = "cartovecto",
                 layer_name = "BDCARTO_BDD_WLD_WGS84:trcon_route")

# Verif
qtm(roads)

## End(Not run)
```

---

**get_wms_info**

Retrieve additional information for wms layer

---

**Description**

`# Usage: get_wms_info(shape, apikey = "ortho", layer_name = "ORTHOIMAGERY.ORTHOPHOTOS.BDORTHO", version = "1.3.0")`
get_wms_raster

Usage

get_wms_info(
    shape,
    apikey = "ortho",
    layer_name = "ORTHOIMAGERY.ORTHOPHOTOS",
    version = "1.3.0"
)

Arguments

shape          Object of class sf. Needs to be located in France.
apikey         API key from get_apikeys() or directly from IGN website
layer_name     Name of the layer from get_layers_metadata(apikey, "wms") or directly from IGN website
version        The version of the service used. More details at IGN documentation

Value

data.frame containing additional information from the layer

Examples

## Not run:
library(sf)

shape <- st_polygon(list(matrix(c(-4.373937, 47.79859,
                                  -4.375615, 47.79738,
                                  -4.375147, 47.79683,
                                  -4.373898, 47.79790,
                                  -4.373937, 47.79859,
                                  ncol = 2, byrow = TRUE)))
shape <- st_sfc(shape, crs = st_crs(4326))

wms_info <- get_wms_info(shape, "ortho", "ORTHOIMAGERY.ORTHOPHOTOS")
date_vol <- wms_info$date_vol

## End(Not run)

dataset_raster

Description

Directly download a raster layer from the French National Institute of Geographic and Forestry. To do that, it need a location giving by a shapefile, an apikey and the name of layer. You can find those information from IGN website
Usage

get_wms_raster(shape,
        apikey = "altimetrie",
        layer_name = "ELEVATION.ELEVATIONGRIDCOVERAGE.HIGHRES",
        resolution = 5,
        filename = NULL,
        crs = 2154,
        overwrite = FALSE,
        version = "1.3.0",
        format = "image/geotiff",
        styles = ""
)

Arguments

- **shape**: Object of class `sf`. Needs to be located in France.
- **apikey**: API key from `get_apikeys()` or directly from IGN website
- **layer_name**: Name of the layer from `get_layers_metadata(apikey, "wms")` or directly from IGN website
- **resolution**: Cell size in meter. WMS are limited to 2048x2048 pixels so depending of the shape and the resolution, correct number and size of tiles is calculated. See detail for more information about resolution.
- **filename**: Either a character string naming a file or a connection open for writing. (ex : "test" or "/test"). The resolution is automatically added to the filename. If raster with same name is already downloaded it is directly imported into R. You don’t have to specify the extension because it is defined in the argument `format`.
- **crs**: Numeric, character, or object of class `sf` or `sfc`. Is set to EPSG:4326 by default. See `sf::st_crs()` for more detail.
- **overwrite**: If TRUE, output raster is overwrite
- **version**: The version of the service used. See detail for more information about `version`.
- **format**: The output format of the image file. Set to geotiff by default. See detail for more information about `format`.
- **styles**: The rendering style of the layers. Set to "" by default. See detail for more information about `styles`.

Details

- Setting the **resolution** parameter higher than the base resolution of the layer multiplies the number of pixels without increasing the precision. For example, the download of the BD Alti layer from IGN will be optimal for a resolution of 25m.
- **version**, **format** and **styles** parameters are detailed on IGN documentation

Value

get_wms_raster return an object of class `stars`. Depending on the layer, this can be a simple raster (2 dimensions and 1 attribute) or an RGB raster (3 dimensions and 1 attribute).
See Also

get_apikeys(), get_layers_metadata(), download.file()

Examples

## Not run:
library(sf)
library(stars)
library(tmap)

apikey <- get_apikeys()[4]

metadata_table <- get_layers_metadata(apikey, "wms")
layer_name <- as.character(metadata_table[2,2])

# shape from the best town in France
shape <- st_polygon(list(matrix(c(-4.373937, 47.79859, 
     -4.375615, 47.79738, 
     -4.375147, 47.79683, 
     -4.373898, 47.79790, 
     -4.373937, 47.79859),
     ncol = 2, byrow = TRUE)))

shape <- st_sfc(shape, crs = st_crs(4326))

# Downloading digital elevation model from IGN
mnt <- get_wms_raster(shape, apikey, layer_name, resolution = 25, filename = "raster_name")
file.remove("raster_name_25m.tif") # Don't want to keep raster on disk
mnt[mnt < 0] <- NA # remove negative values in case of singularity
names(mnt) <- "Elevation [m]" # Rename raster ie the title legend

# Verif
qtm(mnt)+
qtm(shape, fill = NULL, borders.lwd = 3)

## End(Not run)

---

shp_to_geojson

Convert sf or sfc to geojson format

**Description**

Convert sf or sfc to geojson format

**Usage**

shp_to_geojson(shape)

**Arguments**

| shape | A shape of class sf or sfc. Could be a POLYGON, POINT or LINestring. |
Value

   Return a geojson string
Index

* datasets
  cog_2022, 3
are_queryable, 2
cog_2022, 3
download.file(), 15
get_apicarto_cadastre, 3
get_apicarto_commune, 5
get_apicarto_plu, 6
get_apikeys, 8
get_apikeys(), 9, 11, 15
get_layers_metadata, 8
get_layers_metadata(), 11, 15
get_raw_lidar, 10
get_wfs, 11
get_wms_info, 12
get_wms_info(), 2
get_wms_raster, 13

sf::st_crs(), 14
shp_to_geojson, 15