Package 'happign'

November 18, 2022

Title R Interface to 'IGN' Web Services

Version 0.1.7

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, dplyr, geojsonsf, httr2, magrittr, methods, sf (>= 1.0-7), terra, xml2

Suggests covr, httptest2, knitr, rmarkdown, tmap, testthat (>= 3.0.0),

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 2022-11-18 19:10:02 UTC

R topics documented:

`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()`

cog_2022 *COG 2022*

Description

A dataset containing insee code and wording 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

get_apicarto_cadastre *Apicarto Cadastre*

Description

Implementation of the cadastre module of the IGN's apicarto

Usage

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

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

```
## 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))
```

```
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(PCI_shape)+
   tm_borders()+
tm_shape(line)+
   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 for commune borders

Usage

```
get_apicarto_commune(x,
                     source = "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`	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_communereturn` an object of class `sf`

Examples

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

# Using shape
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)

# Using code_insee
commune <- get_apicarto_commune("29158")

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

# Get multiple communes borders

input <- list(line, "29171")
borders <- lapply(input, get_apicarto_commune, source = "PCI")
borders <- do.call(rbind, borders)

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

## End(Not run)
```

---

get_apicarto_gpu *Apicarto module Geoportail de l'urbanisme*

Description

Apicarto module Geoportail de l'urbanisme

Usage

```
get_apicarto_gpu(x,
                 ressource = "zone-urba",
                 partition = NULL,
                 categorie = NULL,
                 dTolerance = 0)
```

Arguments

x	An object of class sf or sfc. If NULL, partition must be filled by partition of GPU.
ressource	A character from this list : "document", "zone-urba", "secteur-cc", "prescription-surf", "prescription-lin", "prescription-pct", "info-surf", "info-lin", "info-pct". See detail for more info.
partition	A character corresponding to GPU partition (can be retrieve using get_apicarto_gpu(x, "document", partition = NULL)). If partition is explicitly set, all GPU features are returned and geom is override
categorie	public utility easement according to the national nomenclature ("http://www.geoinformations.developpem durable.gouv.fr/nomenclature-nationale-des-sup-r1082.html")
dTolerance	To complex shape cannot be handle by API; using dTolerance allow allows to simplify them. See ?sf::st_simplify

Details

For the moment the API cannot returned more than 5000 features.

All resssources description :

- "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, ...
- "acte-sup" :
- "assiette-sup-s" :
- "assiette-sup-l" :
- "assiette-sup-p" :
- "generateur-sup-s" :
- "generateur-sup-l" :
- "generateur-sup-p" :

Value

A object of class `sf`

Examples

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

# If you know the partition, all GPU features are returned, geom is override
partition <- "DU_17345"
poly <- get_apicarto_gpu(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 GPU features is returned)
poly <- get_apicarto_gpu(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_gpu(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)
all_prescription <- map(.x = ressources,
                        .f = ~ get_apicarto_gpu(point, .x, partition))

## End(Not run)
```

`get_apikeys` *List of all API keys from IGN*

Description

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

Usage

```
get_apikeys()
```

Value

character

Examples

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

# All API keys
get_apikeys()


## End(Not run)
```

`get_last_news` *Print latest news from geoservice*

Description

This function connects directly to the RSS feed of the geoservice site to get the latest information. This allows to understand why some resources are sometimes not available.

Usage

```
get_last_news()
```

Value

message

Examples

```
## Not run:
get_last_news()

## 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

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

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.

Value

data.frame

See Also

get_apikeys()

Examples

```
## Not run:
apikey <- get_apikeys()[4]
metadata_table <- get_layers_metadata(apikey, "wms")
all_layer_name <- metadata_table$Name
one_abstract <- metadata_table[1, "Abstract"]

# list 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
list_metadata <- do.call(rbind, list_metadata)

## End(Not run)
```

---

get_raw_lidar *Download raw LIDAR data*

Description

Check if raw LIDAR data are available at the shape location. The raw LIDAR data are not classified; they correspond to a cloud point.

Usage

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

Arguments

shape	Object of class sf or sfc. 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 download 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 and lidar data to destfile. For both directory, function check if grid or data already exist to avoid re-downloading them.

Value

No object.

Examples

```
## 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_raw_lidar(shape)

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

## End(Not run)
```

---

get_wfs *Download WFS layer*

Description

Download a shapefile layer from IGN Web Feature Service (WFS). To do that, it need a location giving by a shape, an apikey and the name of layer. You can find those information from IGN website

Usage

```
get_wfs(shape,
        apikey = "cartovecto",
        layer_name = "BDCARTO_BDD_WLD_WGS84G:troncon_route",
        filename = NULL,
        overwrite = FALSE,
        interactive = FALSE)
```

Arguments

shape	Object of class sf. Needs to be located in France. Bbox of shape is used to intersect features.
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")
overwrite	If TRUE, file is overwrite
interactive	if set to TRUE, no need to specify apikey and layer_name, you'll be ask.

Details

- IGN limits the number of shapes downloaded at the same time to 1000. get_wfs allows to override this limit by making repeated requests but if very large input areas is used (ex : all of France), depending on the resource, this can be time consuming;
- By default, when filename is set, shape are saved as .shp but if names are too long, .gpkg is used.

Value

get_wfsreturn an object of class sf

See Also

get_apikeys(), get_layers_metadata()

Examples

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

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

# For quick testing, use interactive = TRUE
shape_of_my_choice <- get_wfs(shape = shape,
                              interactive = TRUE)

# For specific use, choose apikey with get_apikey() and layer_name with get_layers_metadata()
## Getting borders of best town in France
apikey <- get_apikeys()[1]
metadata_table <- get_layers_metadata(apikey, "wfs")
layer_name <- as.character(metadata_table[32,1])

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

# Plotting result
qtm(borders, fill = NULL, borders = "firebrick") # easy map

## Get forest_area of the best town in France
forest_area <- get_wfs(shape = borders,
                       apikey = "environnement",
                       layer_name = "LANDCOVER.FORESTINVENTORY.V1:resu_bdv1_shape")

qtm(forest_area, fill = "libelle")


## End(Not run)
```

`get_wms_info` *Retrieve additional information for wms layer*

Description

For some wms layer more information can be found with GetFeatureInfo request. This function first check if info are available. If not, available layers are returned.

Usage

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

Arguments

shape	Object of class sf or sfc. 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

Details

#' @usage get_wms_info(shape, apikey = "ortho", layer_name = "ORTHOIMAGERY.ORTHOPHOTOS.BDORTHO", version = "1.3.0"

The function use the centroid of the shape to return info because sometime there's multiple tile.

Value

character 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")

wms_info


## End(Not run)
```

---

get_wms_raster *Download WMS raster layer*

Description

Download a raster layer from IGN Web Mapping Services (WMS). To do that, it need a location giving by a shape, an apikey and the name of layer. You can find those information from IGN website or with get_apikeys() and get_layers_metadata().

Usage

```
get_wms_raster(shape,
               apikey = "altimetrie",
               layer_name = "ELEVATION.ELEVATIONGRIDCOVERAGE.HIGHRES",
               resolution = 5,
               filename = tempfile(fileext = ".tif"),
               crs = 2154,
               overwrite = FALSE,
               version = "1.3.0",
               styles = "",
               interactive = FALSE)
```

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. See detail for more information about resolution.
filename	Either a character string naming a file or a connection open for writing. (ex : "test.tif" or "~/test.tif"). If NULL, layer_name is used. Default drivers is ".tif" but all gdal drivers are supported, see details for more info. To avoid re-downloads, get_wms_raster checks that there is not already a raster with that name. If it does, it is imported into R without further downloading if overwrite is set to FALSE.
crs	Numeric, character, or object of class sf or sfc. It is set to EPSG:2154 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.
styles	The rendering style of the layers. Set to "" by default. See detail for more information about styles.
interactive	If set to TRUE, no need to specify apikey and layer_name, you'll be ask.

Details

- Raster tile are limited to 2048x2048 pixels so depending of the shape and the resolution, correct number of tiles to download is calculated. This mean that setting the resolution argument 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 and styles arguments are detailed on IGN documentation
- Using the crs argument avoids post-reprojection which can be time consuming
- All GDAL supported drivers can be found here

Value

get_wms_raster return an object of class SpatRaster from terra package.

See Also

get_apikeys(), get_layers_metadata()

Examples

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

# 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))

# For quick testing, use interactive = TRUE
raster <- get_wms_raster(shape = shape, interactive = TRUE, filename = tempfile())

# For specific use, choose apikey with get_apikey() and layer_name with get_layers_metadata()
apikey <- get_apikeys()[4]  # altimetrie
metadata_table <- get_layers_metadata(apikey, "wms") # all layers for altimetrie wms
layer_name <- as.character(metadata_table[2,1]) # ELEVATION.ELEVATIONGRIDCOVERAGE

# Downloading digital elevation model from IGN
mnt <- get_wms_raster(shape, apikey, layer_name, resolution = 25, filename = tempfile())

# Preparing raster for plotting
mnt[mnt < 0] <- NA # remove negative values in case of singularity
names(mnt) <- "Elevation [m]" # Rename raster ie the title legend

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

## End(Not run)
```

Index