Package 'happign'

March 28, 2025

Title R Interface to 'IGN' Web Services

Version 0.3.3

Maintainer Paul Carteron <carteronpaul@gmail.com>

Description Automatic open data acquisition from resources of IGN ('Institut National de Information Geographique et forestiere') (<https://www.ign.fr/>). Available datasets include various types of raster and vector data, such as digital elevation models, state borders, spatial databases, cadastral parcels, and more. There also access to point clouds data ('LIDAR') and specifics API (<https://apicarto.ign.fr/api/doc/>).

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 (>= 4.1.0)

Imports archive, dplyr, jsonlite, httr2 (>= 1.1.0), methods, sf (>= 1.0-7), terra, xml2, yyjsonr

Suggests covr, ggplot2, httptest2, knitr, rmarkdown, testthat (>= 3.0.0), tmap (>= 4.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

LazyData true

NeedsCompilation no

Encoding UTF-8

RoxygenNote 7.3.2

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 2025-03-28 12:50:05 UTC

Contents

`are_queryable` *are_queryable*

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_location_info()`

build_iso_query *build_iso_query*

Description

build query for isochrone-dist API

Usage

```
build_iso_query(
  point,
  source,
  value,
  type,
  profile,
  direction,
  constraints,
  distance_unit,
  time_unit
)
```

Arguments

point	character; point formated with x_to_iso.
source	character; This parameter specifies which source will be used for the calculation. Currently, "valhalla" and "pgr" sources are available (default "pgr"). See section SOURCE for further information.
value	numeric; A quantity of time or distance.
type	character; Specifies the type of calculation performed: "time" for isochrone or "distance" for isodistance (isochrone by default).
profile	character; Type of cost used for calculation: "pedestrian" for #' pedestrians and "car" for cars. and "car" for cars ("pedestrian" by default).
direction	character; Direction of travel. Either define a "departure" point and obtain the potential arrival points. Or define an "arrival" point and obtain the potential points ("departure" by default).
constraints	Used to express constraints on the characteristics to calculate isochrones/isodistances. See section CONSTRAINTS.
distance_unit	character; Allows you to specify the unit in which distances are expressed in the answer: "meter" or "kilometer" (meter by default).
time_unit	character; Allows you to specify the unit in which times are expressed in the answer: "hour", "minute" or "second" (minutes by default).

Value

httr2_request object

cog_2023 *COG 2023*

Description

A dataset containing insee code and wording of commune as of January 1, 2023. COG mean Code Officiel Géographique

Usage

```
cog_2023
```

Format

`cog_2023`:

A data frame with 34990 rows and 2 columns:

COM insee code

LIBELLE Name of commune

Source

https://www.insee.fr/fr/information/6800675

com_2024 *COG 2024*

Description

A dataset containing insee code and wording of commune as of January 1, 2024. COG mean Code Officiel Géographique

Usage

```
com_2024
```

Format

`com_2024`:

A data frame with 34980 rows and 2 columns:

COM insee code

LIBELLE Name of commune

Source

https://www.insee.fr/fr/information/7766585

`get_apicarto_cadastre` *Apicarto Cadastre*

Description

Implementation of the cadastre module from the IGN's apicarto

Usage

```
get_apicarto_cadastre(x,
                      type = "commune",
                      code_com = NULL,
                      section = NULL,
                      numero = NULL,
                      code_arr = NULL,
                      code_abs = NULL,
                      dTolerance = 0L,
                      source = "pci",
                      progress = TRUE)
```

Arguments

`x`	It can be a shape, insee codes or departement codes : • Shape : must be an object of class `sf` or `sfc`. • Code insee : must be a `character` of length 5 • Code departement : must be a `character` of length 2 or 3 (DOM-TOM)
`type`	A `character` from `"parcelle"`, `"commune"`, `"feuille"`, `"division"`, `"localisant"`
`code_com`	A `character` of length 5 corresponding to the commune code. Only use with `type = "division"` or `type = "feuille"`
`section`	A `character` of length 2
`numero`	A `character` of length 4
`code_arr`	A `character` corresponding to district code for Paris, Lyon, Marseille
`code_abs`	A `character` corresponding to the code of absorbed commune. This prefix is useful to differentiate between communes that have merged
`dTolerance`	numeric; Complex shape cannot be handle by API; using `dTolerance` allow to simplify them. See `?sf::st_simplify`
`source`	Can be "bdp" for BD Parcellaire or "pci" for Parcellaire express. See detail for more info.
`progress`	Display a progress bar? Use TRUE to turn on a basic progress bar, use a string to give it a name. See `httr2::req_perform_iterative()`.

Details

`x`, `section`, `numero`, `code_arr`, `code_abs`, `code_com` can take vector of character. In this case vector recycling is done. See the example section below.

`source`: 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

Object of class `sf`

Examples

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

# shape from the town of penmarch
penmarch <- read_sf(system.file("extdata/penmarch.shp", package = "happign"))

# get commune borders
## from shape
penmarch_borders <- get_apicarto_cadastre(penmarch, type = "commune")
qtm(penmarch_borders)+qtm(penmarch, fill = "red")

## from insee_code
border <- get_apicarto_cadastre("29158", type = "commune")
borders <- get_apicarto_cadastre(c("29158", "29135"), type = "commune")
qtm(borders, fill="nom_com")

# get cadastral parcels
## from shape
parcels <- get_apicarto_cadastre(penmarch, type = "parcelle")

## from insee code
parcels <- get_apicarto_cadastre("29158", type = "parcelle")

# Use parameter recycling
## get sections "AW" parcels from multiple insee_code
parcels <- get_apicarto_cadastre(
   c("29158", "29135"),
   section = "AW",
   type = "parcelle"
)
qtm(borders, fill = NA)+qtm(parcels)

## get parcels numbered "0001", "0010" of section "AW" and "BR"
section <- c("AW", "BR")
numero <- rep(c("0001", "0010"), each = 2)
parcels <- get_apicarto_cadastre("29158", section = section, numero = numero, type = "parcelle")
qtm(penmarch_borders, fill = NA)+qtm(parcels)
```

```
## generalization with expand.grid
params <- expand.grid(code_insee = c("29158", "29135"),
                      section = c("AW", "BR"),
                      numero = c("0001", "0010"),
                      stringsAsFactors = FALSE)
parcels <- get_apicarto_cadastre(params$code_insee,
                                 section = params$section,
                                 numero = params$numero,
                                 type = "parcelle")
qtm(penmarch_borders, fill = NA)+qtm(parcels$geometry)


## End(Not run)
```

`get_apicarto_codes_postaux` *Apicarto Codes Postaux*

Description

Implementation of the "Codes Postaux" module from the IGN's apicarto. This API give information about commune from postal code.

Usage

```
get_apicarto_codes_postaux(code_post)
```

Arguments

`code_post`	`character` corresponding to the postal code of a commune

Value

Object of class `data.frame`

Examples

```
## Not run:

info_commune <- get_apicarto_codes_postaux("29760")

code_post <- c("29760", "29260")
info_communes <- get_apicarto_codes_postaux(code_post)

## 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",
                 categorie = list(NULL),
                 dTolerance = 0)
```

Arguments

`x`	An object of class `sf` or `sfc` for geometric intersection. Otherwise a `character` corresponding to **GPU partition** or **insee code** when `ressource` is set to `municipality`.
`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.
`categorie`	public utility easement according to the national nomenclature
`dTolerance`	numeric; Complex shape cannot be handle by API; using `dTolerance` allow to simplify them. See `?sf::st_simplify`

Details

/!\ For the moment the API cannot returned more than 5000 features.

All existing parameters for `ressource` :

- "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", "prescription-lin", "prescription-pct" : its's a constraint or a possibility indicated in an urban planning document (PLU, PLUi, ...)
- "info-surf", "info-lin", "info-pct" : its's an information indicated in an urban planning document (PLU, PLUi, ...)
- "acte-sup" : act establishing the SUP
- "generateur-sup-s", "generateur-sup-l", "generateur-sup-p" : an entity (site or monument, watercourse, water catchment, electricity or gas distribution of electricity or gas, etc.) which generates on the surrounding SUP (of passage, alignment, protection, land reservation, etc.)
- "assiette-sup-s", "assiette-sup-l", "assiette-sup-p" : spatial area to which SUP it applies.

Value

A object of class `sf` or `df`

Examples

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

# find if commune is under the RNU (national urbanism regulation)
rnu <- get_apicarto_gpu("93014", "municipality")
rnu$is_rnu

# get urbanism document
x <- get_apicarto_cadastre("93014", "commune")
document <- get_apicarto_gpu(x, ressource = "document")
partition <- document$partition

# get gpu features
## from shape
gpu <- get_apicarto_gpu(x, ressource = "zone-urba")

## from partition
gpu <- get_apicarto_gpu("DU_93014", ressource = "zone-urba")

# example : all prescriptions
ressources <- c("prescription-surf",
                "prescription-lin",
                "prescription-pct")
prescriptions <- get_apicarto_gpu("DU_93014",
                                  ressource = ressources)

# example : public utility servitude (SUP) assiette
assiette_sup_s <- get_apicarto_gpu(x, ressource = "assiette-sup-s")
protection_forest <- get_apicarto_gpu(x,
                                      ressource = "assiette-sup-s",
                                      categorie = "A7")

# example : public utility servitude (SUP) generateur
## /!\ a generator can justify several assiette
ressources <- c("generateur-sup-p",
                "generateur-sup-l",
                "generateur-sup-s")
all_gen <- get_apicarto_gpu(x, ressource = ressources)


## End(Not run)
```

`get_apicarto_rpg` *Apicarto RPG (Registre Parcellaire Graphique)*

Description

Implementation of the "RPG" module from the IGN's apicarto. This function is a wrapper around version 1 and 2 of the API.

Usage

```
get_apicarto_rpg(x,
                 annee,
                 code_cultu = NULL,
                 dTolerance = 0L,
                 progress = FALSE)
```

Arguments

x	Object of class sf. Needs to be located in France.
annee	numeric between 2010 and 2023.
code_cultu	character corresponding to code culture, see detail.
dTolerance	numeric; tolerance parameter. The value of dTolerance must be specified in meters, see detail.
progress	Display a progress bar? Use TRUE to turn on a basic progress bar, use a string to give it a name. See `httr2::req_perform_iterative()`.

Details

Since 2014 the culture code has changed its format. Before it should be a value ranging from "01" to "28", after it should be a trigram (ex : "MIE"). More info can be found at the documentation page

dTolerance is needed when geometry are too complex. Its the same parameter found in `sf::st_simlplify`.

Value

list or object of class sf

Examples

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

penmarch <- get_apicarto_cadastre("29158", type = "commune")

# failure with too complex geom
rpg <- get_apicarto_rpg(st_buffer(penmarch, 10), 2020)

# avoid complex data by setting dTolerance
rpg <- get_apicarto_rpg(penmarch, 2020, dTolerance = 15)

# multiple years after 2014
rpg <- get_apicarto_rpg(penmarch, 2020:2021, dTolerance = 15)
```

```
# years before and after 2014
# list is returned because attributs are different
rpg <- get_apicarto_rpg(penmarch, c(2010, 2021), dTolerance = 15)

# filter by code_cultu
rpg <- get_apicarto_rpg(penmarch, 2021, code_cultu = "MIE", dTolerance = 15)

# all "MIE" from 2020 and all "PPH" from 2021
rpg <- get_apicarto_rpg(penmarch, 2020:2021,
                        code_cultu = c("MIE", "PPH"),
                        dTolerance = 15)

# vectorization : all "MIE" from 2020 and 2021
rpg <- get_apicarto_rpg(x, 2020:2021, code_cultu = "MIE", dTolerance = 15)

## End(Not run)
```

`get_apicarto_viticole` *Apicarto Appellations viticoles*

Description

Implementation of the "Appellations viticoles" module from the IGN's apicarto. The module uses a database maintained by FranceAgriMer. This database includes : appellation d'origine contrôlée (AOC) areas, protected geographical indication areas (IGP) and wine growing areas without geographical indications (VSIG)

Usage

```
get_apicarto_viticole(x,
                      dTolerance = 0)
```

Arguments

`x`	Object of class `sf`. Needs to be located in France.
`dTolerance`	numeric; tolerance parameter. The value of `dTolerance` must be specified in meters, see `?sf::st_simplify` for more info.

Details

/!\ For the moment the API cannot returned more than 1000 features.

Value

Object of class `sf`

Examples

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

penmarch <- read_sf(system.file("extdata/penmarch.shp", package = "happign"))

VSIG <- get_apicarto_viticole(penmarch)

## 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_iso *isochronous/isodistance calculations*

Description

Calculates isochrones or isodistances in France from an sf object using the IGN API on the Géoportail platform. The reference data comes from the IGN BD TOPO® database. For further information see IGN documentation.

Usage

```
get_iso(x,
        value,
        type = "time",
        profile = "pedestrian",
        time_unit = "minute",
        distance_unit = "meter",
        direction = "departure",
        source = "pgr",
        constraints = NULL)

get_isodistance(x,
                dist,
                unit = "meter",
                source = "pgr",
                profile = "car",
                direction = "departure",
                constraints = NULL)

get_isochrone(x,
              time,
              unit = "minute",
              source = "pgr",
              profile = "car",
              direction = "departure",
              constraints = NULL)
```

Arguments

x	Object of class sf or sfc with POINT geometry. There may be several points in the object. In this case, the output will contain as many polygons as points.
value	numeric; A quantity of time or distance.
type	character; Specifies the type of calculation performed: "time" for isochrone or "distance" for isodistance (isochrone by default).
profile	character; Type of cost used for calculation: "pedestrian" for #' pedestrians and "car" for cars. and "car" for cars ("pedestrian" by default).

time_unit	character; Allows you to specify the unit in which times are expressed in the answer: "hour", "minute" or "second" (minutes by default).
distance_unit	character; Allows you to specify the unit in which distances are expressed in the answer: "meter" or "kilometer" (meter by default).
direction	character; Direction of travel. Either define a "departure" point and obtain the potential arrival points. Or define an "arrival" point and obtain the potential points ("departure" by default).
source	character; This parameter specifies which source will be used for the calculation. Currently, "valhalla" and "pgr" sources are available (default "pgr"). See section SOURCE for further information.
constraints	Used to express constraints on the characteristics to calculate isochrones/isodistances. See section CONSTRAINTS.
dist	numeric; A quantity of time.
unit	see time_unit and distance_unit param.
time	numeric; A quantity of time.

Value

object of class sf with POLYGON geometry

Functions

- get_isodistance(): Wrapper function to calculate isodistance from get_iso.
- get_isochrone(): Wrapper function to calculate isochrone from get_iso.

SOURCE

Isochrones are calculated using the same resources as for route calculation. PGR" and "VALHALLA" resources are used, namely "bdtopo-valhalla" and "bdtopo-pgr".

- bdtopo-valhalla" : To-Do
- bdtopo-iso" is based on the old services over a certain distance, to solve performance problems. We recommend its use for large isochrones.

PGR resources are resources that use the PGRouting engine to calculate isochrones. ISO resources are more generic. The engine used for calculations varies according to several parameters. At present, the parameter concerned is cost_value, i.e. the requested time or distance.

See Also

get_isodistance, get_isochrone

Examples

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

# All area i can acces in less than 5 minute from penmarch centroid
penmarch <- get_apicarto_cadastre("29158")
penmarch_centroid <- st_centroid(penmarch)
isochrone <- get_isochrone(penmarch_centroid, 5)

qtm(penmarch, col = "red")+qtm(isochrone, col = "blue")+qtm(penmarch_centroid, fill = "red")

# All area i can acces as pedestrian in less than 1km
isodistance <- get_isodistance(penmarch_centroid, 1, unit = "kilometer", profile = "pedestrian")

qtm(penmarch, col = "red")+qtm(isodistance, col = "blue")+qtm(penmarch_centroid, fill = "red")

# In case of multiple point provided, the output will contain as many polygons as points.
code_insee <- c("29158", "29072", "29171")
communes_centroid <- get_apicarto_cadastre(code_insee) |> st_centroid()
isochrones <- get_isochrone(communes_centroid, 8)
isochrones$code_insee <- code_insee
qtm(isochrones, fill = "code_insee")

# Find area where i can acces all communes centroid in less than 8 minutes
area <- st_intersection(isochrones)
qtm(communes_centroid, fill = "red")+ qtm(area[area$origins == "1:3",])

## End(Not run)
```

`get_last_news` *Print latest news from geoservice website*

Description

This function is a wrapper around the RSS feed of the geoservice site to get the latest information.

Usage

```
get_last_news()
```

Value

message or error

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(data_type, apikey = NULL)
```

Arguments

`data_type`	Should be `"wfs"`, `"wms-r"` or `"wmts"`. See details for more information about these Web services formats.
`apikey`	API key from `get_apikeys()` or directly from the IGN website

Details

- `"wfs"` : Web Feature Service designed to return data in vector format (line, point, polygon, ...) ;
- `"wms-r"` : Web Map Service focuses on raster data ;
- `"wmts"` : Web Map Tile Service is similar to WMS, but instead of serving maps as single images, WMTS serves maps by dividing the map into a pyramid of tiles at multiple scales.

Value

data.frame

See Also

`get_apikeys()`

Examples

```
## Not run:
# Get all metadata for a datatype
metadata_table <- get_layers_metadata("wms-r")

# Get all "administratif" wms layers
apikey <- get_apikeys()[1] #administratif
admin_layers <- get_layers_metadata("wms-r", apikey)


## End(Not run)
```

`get_location_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_location_info(x,
                  apikey = "ortho",
                  layer = "ORTHOIMAGERY.ORTHOPHOTOS",
                  read_sf = TRUE,
                  version = "1.3.0")
```

Arguments

`x`	Object of class `sf` or `sfc`. Only single point are supported for now. Needs to be located in France.
`apikey`	`character`; API key from get_apikeys() or directly from the IGN website
`layer`	`character`; layer name obtained from `get_layers_metadata("wms-r")` or the IGN website.
`read_sf`	`logical`; if `TRUE` an `sf` object is returned but response times may be higher.
`version`	`character`; old param

Value

`character` or `sf` containing additional information about the layer

Examples

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

# From single point
x <- st_centroid(read_sf(system.file("extdata/penmarch.shp", package = "happign")))
location_info <- get_location_info(x, "ortho", "ORTHOIMAGERY.ORTHOPHOTOS", read_sf = F)
location_info$date_vol

# From multiple point
x1 <- st_sfc(st_point(c(-3.549957, 47.83396)), crs = 4326) # Carnoet forest
x2 <- st_sfc(st_point(c(-3.745995, 47.99296)), crs = 4326) # Coatloch forest

forests <- lapply(list(x1, x2),
                  get_location_info,
                  apikey = "environnement",
                  layer = "FORETS.PUBLIQUES",
                  read_sf = T)

qtm(forests[[1]]) + qtm(forests[[2]])

# Find all queryable layers
queryable_layers <- lapply(get_apikeys(), are_queryable) |> unlist()

## 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(x, destfile = ".", grid_path = ".", quiet = F)
```

Arguments

x	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 `x` 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
x <- 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)))
x <- st_sfc(x, crs = st_crs(4326))

# Download data to current directory
get_raw_lidar(x)

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

## End(Not run)
```

`get_wfs` *Download WFS layer*

Description

Read simple features from IGN Web Feature Service (WFS) from location and name of layer.

Usage

```
get_wfs(x = NULL,
        layer = NULL,
        filename = NULL,
        spatial_filter = "bbox",
        ecql_filter = NULL,
        overwrite = FALSE,
        interactive = FALSE)
```

Arguments

x	Object of class sf or sfc. Needs to be located in France.
layer	character; name of the layer from get_layers_metadata("wfs") or directly from IGN website
filename	character;Either a string naming a file or a connection open for writing. (ex : "test.shp" or "~/test.shp")
spatial_filter	character; spatial predicate from ECQL language. See detail and examples for more info.
ecql_filter	character; corresponding to an ECQL query. See detail and examples for more info.
overwrite	logical; if TRUE, file is overwrite.
interactive	character; if TRUE, no need to specify layer, you'll be ask.

Details

- get_wfs use ECQL language : a query language created by the OpenGeospatial Consortium. It provide multiple spatial filter : "intersects", "disjoint", "contains", "within", "touches", "crosses", "overlaps", "equals", "relate", "beyond", "dwithin". For "relate", "beyond", "dwithin", argument can be provide using vector like : spatial_filter = c("dwithin", distance, units). More info about ECQL language here.
- ECQL query can be provided to ecql_filter. This allows direct query of the IGN's WFS geoservers. If x is set, then the ecql_filter comes in addition to the spatial_filter. More info for writing ECQL here

Value

sf object from sf package or NULL if no data.

See Also

get_layers_metadata()

Examples

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

# Shape from the best town in France
penmarch <- read_sf(system.file("extdata/penmarch.shp", package = "happign"))

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

## Getting borders of best town in France
metadata_table <- get_layers_metadata("wfs", "administratif")
layer <- metadata_table[32,1] # LIMITES_ADMINISTRATIVES_EXPRESS.LATEST:commune
```

```
# Downloading borders
borders <- get_wfs(penmarch, layer)

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

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

qtm(forest_area, fill = "nom_typn")

# Using ECQL filters to query IGN server
## First find attributes of the layer
attrs <- get_wfs_attributes(layer)

## e.g. : find all commune's name starting by "plou"
plou_borders <- get_wfs(x = NULL, # When x is NULL, all France is query
                        layer = "LIMITES_ADMINISTRATIVES_EXPRESS.LATEST:commune",
                        ecql_filter = "nom_m LIKE 'PLOU%'")
qtm(plou_borders)

## Combining ecql_filters
plou_borders_inf_2000 <- get_wfs(x = NULL, # When x is NULL, all France is query
                                 layer = "LIMITES_ADMINISTRATIVES_EXPRESS.LATEST:commune",
                                 ecql_filter = "nom_m LIKE 'PLOU%' AND population < 2000")
qtm(plou_borders)+ qtm(plou_borders_inf_2000, fill = "red")

## End(Not run)
```

`get_wfs_attributes` *get_wfs_attributes*

Description

Helper to write ecql filter. Retrieve all attributes from a layer.

Usage

```
get_wfs_attributes(layer = NULL, interactive = FALSE)
```

Arguments

`layer`	`character`; name of the layer from `get_layers_metadata("wfs")` or directly from IGN website
`interactive`	`character`; if TRUE, no need to specify `layer`, you'll be ask.

Value

charactervector with layer attributes

Examples

```
## Not run:

get_wfs_attributes("administratif", "LIMITES_ADMINISTRATIVES_EXPRESS.LATEST:commune")

# Interactive session
get_wfs_attributes(interactive = TRUE)


## End(Not run)
```

---

get_wms_raster *Download WMS raster layer*

Description

Download a raster layer from the IGN Web Mapping Services (WMS). Specify a location using a shape and provide the layer name.

Usage

```
get_wms_raster(x,
               layer = "ORTHOIMAGERY.ORTHOPHOTOS",
               res = 10,
               crs = 2154,
               rgb = TRUE,
               filename = NULL,
               overwrite = FALSE,
               verbose = TRUE,
               interactive = FALSE)
```

Arguments

x	Object of class sf or sfc, located in France.
layer	character; layer name obtained from get_layers_metadata("wms-r") or the IGN website.
res	numeric; resolution specified in the units of the coordinate system (e.g., meters for EPSG:2154, degrees for EPSG:4326). See details for more information.
crs	numeric, character, or object of class sf or sfc; defaults to EPSG:2154. See sf::st_crs() for more details.
rgb	boolean; if set to TRUE, downloads an RGB image. If set to FALSE, downloads a single band with floating point values. See details for more information.

filename	character or NULL; specifies the filename or an open connection for writing (e.g., "test.tif" or "~/test.tif"). The default format is ".tif" but all GDAL drivers are supported. When a filename is provided, the function uses it as a cache: if the file already exists and overwrite is set to FALSE, the function will directly load the raster from that file instead of re-downloading it.
overwrite	boolean; if TRUE, the existing raster will be overwritten.
verbose	boolean; if TRUE, message are added.
interactive	logical; if TRUE, an interactive menu prompts for apikey and layer argument.

Details

- res: Note that setting res higher than the default resolution of the layer will increase the number of pixels but not the precision of the image. For instance, downloading the BD Alti layer from IGN is optimal at a resolution of 25m.
- rgb: Rasters are commonly used to download images such as orthophotos. In specific cases like DEMs, however, a value per pixel is essential. See examples below.

Value

SpatRaster object from terra package.

See Also

get_layers_metadata()

Examples

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

# Shape from the best town in France
penmarch <- read_sf(system.file("extdata/penmarch.shp", package = "happign"))

# For quick testing use interactive = TRUE
raster <- get_wms_raster(x = penmarch, res = 25, interactive = TRUE)

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

# Downloading digital elevation model values not image
mnt_2154 <- get_wms_raster(penmarch, layer, res = 1, crs = 2154, rgb = FALSE)

# If crs is set to 4326, res is in degrees
mnt_4326 <- get_wms_raster(penmarch, layer, res = 0.0001, crs = 4326, rgb = FALSE)

# Plotting result
```

```
tm_shape(mnt_4326)+
   tm_raster()+
tm_shape(penmarch)+
   tm_borders(col = "blue", lwd  = 3)

## End(Not run)
```

`get_wmts` *Download WMTS raster tiles*

Description

Download an RGB raster layer from IGN Web Map Tile Services (WMTS). WMTS focuses on performance and can only query pre-calculated tiles.

Usage

```
get_wmts(x,
         layer = "ORTHOIMAGERY.ORTHOPHOTOS",
         zoom = 10L,
         crs = 2154,
         filename = tempfile(fileext = ".tif"),
         verbose = FALSE,
         overwrite = FALSE,
         interactive = FALSE)
```

Arguments

`x`	Object of class `sf` or `sfc`. Needs to be located in France.
`layer`	`character`; layer name from `get_layers_metadata(apikey, "wms")` or directly from IGN website.
`zoom`	`integer` between 0 and 21; at low zoom levels, a small set of map tiles covers a large geographical area. In other words, the smaller the zoom level, the less precise the resolution. For conversion between zoom level and resolution see WMTS IGN Documentation
`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.
`filename`	`character` or `NULL`; filename or a open connection for writing. (ex : "test.tif" or "~/test.tif"). If `NULL`, `layer` is used as filename. Default drivers is ".tif" but all gdal drivers are supported, see details for more info.
`verbose`	`boolean`; if TRUE, message are added.
`overwrite`	If TRUE, output raster is overwrite.
`interactive`	`logical`; If TRUE, interactive menu ask for `apikey` and `layer`.

Value

SpatRaster object from terra package.

See Also

get_apikeys(), get_layers_metadata()

Examples

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

penmarch <- read_sf(system.file("extdata/penmarch.shp", package = "happign"))

# Get orthophoto
layers <- get_layers_metadata("wmts", "ortho")$Identifier
ortho <- get_wmts(penmarch, layer = layers[1], zoom = 21)
plotRGB(ortho)

# Get all available irc images
layers <- get_layers_metadata("wmts", "orthohisto")$Identifier
irc_names <- grep("irc", layers, value = TRUE, ignore.case = TRUE)

irc <- lapply(irc_names, function(x) get_wmts(penmarch, layer = x, zoom = 18)) |>
   setNames(irc_names)

# remove empty layer (e.g. only NA)
irc <- Filter(function(x) !all(is.na(values(x))), irc)

# plot
all_plots <- lapply(irc, plotRGB)


## End(Not run)
```

Index