Package 'happign'

February 19, 2026

Title R Interface to 'IGN' Web Services

Version 0.3.8

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. 'happign' also
provide access to API Carto (<https://apicarto.ign.fr/api/doc/>).

License GPL (>= 3)

URL https://paul-carteron.github.io/happign/,
https://github.com/paul-carteron/happign

BugReports https://github.com/paul-carteron/happign/issues

Depends R (>= 4.1.0)

Imports jsonlite, httr2 (>= 1.1.0), sf (>= 1.0-7), terra, xml2

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

VignetteBuilder knitr

Config/testthat/edition 3

LazyData true

NeedsCompilation no

Encoding UTF-8

RoxygenNote 7.3.3

Config/Needs/website rmarkdown

Author Paul Carteron [aut, cre] (ORCID:
<https://orcid.org/0000-0002-6942-6662>)

Repository CRAN

Date/Publication 2026-02-19 06:10:14 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

`com_2025` *French Communes Table (2025)*

Description

Data for French communes from the INSEE file "v_commune_2025.csv".

Usage

```
com_2025
```

Format

A data frame with one row per commune and the following columns:

TYPECOM (chr) Type of commune (4 characters)

COM (chr) Commune code (5 characters)

REG (int) Region code (2 characters)

DEP (chr) Department code (3 characters)

CTCD (chr) Code of the territorial collectivity with departmental powers (4 characters)

ARR (chr) District (arrondissement) code (4 characters)

TNCC_COM (int) Name type indicator (1 character)

NCC_COM (chr) Official name in uppercase (200 characters)

NCCENR_COM (chr) Official name with proper typography (200 characters)

LIBELLE_COM (chr) Official name with article and proper typography (200 characters)

CAN (chr) Canton code (5 characters). For “multi-canton” communes, code ranges from 99 to 90 (pseudo-canton) or 89 to 80 (new communes)

COMPARENT (int) Parent commune code for municipal districts and associated or delegated communes (5 characters)

Source

https://www.insee.fr/fr/statistiques/fichier/8377162/cog_ensemble_2025_csv.zip

`dep_2025` *French Departments Table (2025)*

Description

Data for French departments from the INSEE file "Départements".

Usage

```
dep_2025
```

Format

A data frame with one row per department and the following columns:

DEP (chr) Department code (3 characters)

REG (int) Region code (2 characters)

CHEFLIEU_DEP (chr) Commune code of the departmental capital (5 characters)

TNCC_DEP (int) Name type indicator (1 character)

NCC_DEP (chr) Official name in uppercase (200 characters)

NCCENR_DEP (chr) Official name with proper typography (200 characters)

LIBELLE_DEP (chr) Official name with article and proper typography (200 characters)

Source

`https://www.insee.fr/fr/statistiques/fichier/8377162/cog_ensemble_2025_csv.zip`

`get_apicarto_cadastre` *Apicarto Cadastre*

Description

Implementation of the cadastre module from the IGN's apicarto

Usage

```
get_apicarto_cadastre(x,
                      type = "commune",
                      section = NULL,
                      numero = NULL,
                      code_abs = NULL,
                      source = "pci",
                      progress = TRUE)
```

Arguments

`x`	`sf`, `sfc`, `character` or `numeric` : • Shape : must be an object of class `sf` or `sfc`. • Code insee : must be a `character` of length 5 (see com_2025) • Code departement : must be a `character` of length 2 or 3 (DOM-TOM) (see dep_2025)
`type`	`character` : type of data needed, default to `"commune"`. One of `"commune"`, `"parcelle"`, `"section"`, `"localisant"`.
`section`	`character` : corresponding to section of a city.
`numero`	`character` : corresponding to numero of cadastral parcels.
`code_abs`	`character` : corresponding to the code of absorbed commune. This prefix is useful to differentiate between communes that have merged
`source`	`character` : `"bdp"` for BD Parcellaire or `"pci"` for Parcellaire express. Default to `"pci"`. 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

Vectorisation:

Arguments `x`, `section`, `numero`, and `code_abs` are vectorized if only one argument has `length` > 1 (**Cartesian product**)

```
x = 29158; section = c("A", "B")
→ (29158, "A"), (29158, "B")

x = 29158, section = "A", numero = 1:3
→ (29158, "A", 1); (29158, "A", 2); (29158, "A", 3)
```

In case all vectorised arguments have the same length **Pairwise matching** is used

```
x = c(29158, 29158); section = c("A", "B"); numero = 1:2
→ (29158, "A", 1), (29158, "B", 2)
```

Ambiguous vectorisation:

If more than one argument has `length` > 1 but lengths differ, it is unclear whether to combine them pairwise or via cartesian product. This is rejected with an error to avoid unintended queries.

```
x = 29158, section = c("A", "B"), numero = 1:2
Possible interpretations:
1. Pairwise: (29158, "A", 1), (29158, "B", 2)
2. Cartesian: (29158, "A", 1), (29158, "A", 2), (29158, "B", 1), (29158, "B", 2)
```

Source:

BD Parcellaire (`"bdp"`) is no longer updated and its use is discouraged. PCI Express (`"pci"`) is strongly recommended and will become mandatory. See IGN's product comparison table.

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")
qtm(parcels, fill="section")

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

# 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)

## if multiple args with length > 1 result is ambigous
parcels <- get_apicarto_cadastre(
   x = c("29158", "29135"),
   section = c("AW", "AB"),
   numero = 1,
   type = "parcelle"
)

## get parcels numbered "0001", "0010" of section "AW" and "BR"
insee <- rep("29158", 2)
section <- c("AW", "BR")
numero <- c("0001", "0010")
```

```
parcels <- get_apicarto_cadastre(insee, section = section, numero = numero, type = "parcelle")
qtm(penmarch_borders, fill = NA)+qtm(parcels)

# Arrondissement insee code should be used for paris, lyon, marseille
error <- get_apicarto_cadastre(c(75056, 69123, 13055))
paris_arr123 <- get_apicarto_cadastre(c(75101, 75102, 75103))
qtm(paris_arr123, fill = "code_insee")


## 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", "08170")
info_communes <- get_apicarto_codes_postaux(code_post)

code_post <- c("12345")
info_communes <- get_apicarto_codes_postaux(code_post)

code_post <- c("12345", "08170")
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, layer, category = NULL)
```

Arguments

`x`	`sf`, `sfc` or `character` : • Shape : must be an object of class `sf` or `sfc`. • Code insee (layer = `"municipality"`) : must be a `character` of length 5 (see com_2025) • Partition : must be a valid partition `character` for checking and Geoportail for documentation
`layer`	`character`; Layer name from `get_gpu_layers()`
`category`	public utility easement according to the national nomenclature

Details

/!\ API cannot returned more than 5000 features.

All existing parameters for `layer` :

- `"municipality"` : information on the communes (commune with RNU, merged commune)
- `"document"` : information on urban planning documents (POS, PLU, PLUi, CC, PSMV, SCoT)
- `"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

sf

Examples

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

# Find if commune is under the RNU (national urbanism regulation)
# If no RNU it means communes probably have a PLU
rnu <- get_apicarto_gpu("29158", "municipality")
rnu$is_rnu

# Get urbanism document
# Rq : when using geometry, multiple documents can be returned due to intersection
x <- get_apicarto_cadastre("29158", "commune")
document <- get_apicarto_gpu(x, "document")
document$partition
penmarch <- document$partition[2]

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

## from partition
gpu <- get_apicarto_gpu(penmarch, "zone-urba")
qtm(gpu, fill="typezone")

# example : all prescription
layers <- names(get_gpu_layers("prescription"))
prescriptions <- lapply(layers, \(x) get_apicarto_gpu(penmarch, x)) |>
   setNames(layers)

qtm(prescriptions$`prescription-pct`, fill = "libelle")+
qtm(prescriptions$`prescription-lin`, col = "libelle")+
qtm(prescriptions$`prescription-surf`, fill = "libelle")

# When using SUP, category can be used for filtering
# AC1 : Monuments historiques
penmarch <- get_apicarto_cadastre(29158)
mh <- get_apicarto_gpu(penmarch, "assiette-sup-s", category = "AC1")

# example : public utility servitude (SUP) generateur
## /!\ a generator can justify several assiette
gen_mh <- get_apicarto_gpu(penmarch, "generateur-sup-s", "AC1")

qtm(mh, fill = "lightblue")+qtm(gen_mh, fill = "red")


## 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_gpu_layers` *Available GPU layers*

Description

Helpers that return available GPU layers and their type.

Usage

```
get_gpu_layers(type = NULL)
```

Arguments

`type`	`character` One of `"commune"`, `"du"`, `"prescription"`, `"acte-sup"`, `"assiette"`, `"generateur"`. If `NULL`, all layers are retuned. `NULL` by default

Details

`"du"`: "Document d'urbanisme" `"sup"`: "Servitude d'utilité publique"

Value

list

Examples

```
# All layers
names(get_gpu_layers())

# All sup layers
names(get_gpu_layers("generateur"))

# All sup and du layers
names(get_gpu_layers(c("generateur", "prescription")))
```

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_wfs` *Download data from IGN WFS layer*

Description

Download features from the IGN Web Feature Service (WFS) using a spatial predicate, an ECQL attribute query, or both.

Usage

```
get_wfs(
  x = NULL,
  layer = NULL,
  predicate = bbox(),
  query = NULL,
  verbose = TRUE
)
```

Arguments

`x`	`sf`, `sfc` or `NULL`. If `NULL`, no spatial filter is applied and `query` must be provided.

`layer`	`character`; name of the WFS layer. Must correspond to a layer available on the IGN WFS service (see `get_layers_metadata()`).
`predicate`	`list`; a spatial predicate definition created with helper such as `bbox()`, `intersects()`, `within()`, `contains()`, `touches()`, `crosses()`, `overlaps()`, `equals()`, `dwithin()`, `beyond()` or `relate()`. See spatial_predicates for more info.
`query`	`character`; an ECQL attribute query. When both `x` and `query` are provided, the spatial predicate and the attribute query are combined.
`verbose`	`logical`; if `TRUE`, display progress information and other informative message.

Details

- `get_wfs` use ECQL language : a query language created by the OpenGeospatial Consortium. More info about ECQL language can be found here.

Value

An object of class `sf`.

See Also

`get_layers_metadata()`

Examples

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

# Load a geometry
x <- read_sf(system.file("extdata/penmarch.shp", package = "happign"))

# Retrieve commune boundaries intersecting x
commune <- get_wfs(
  x = x,
  layer = "LIMITES_ADMINISTRATIVES_EXPRESS.LATEST:commune"
)

plot(st_geometry(commune), border = "firebrick")

# Attribute-only query (no spatial filter)

# If unknown, available attributes can be retrieved using `get_wfs_attributes()`
attrs <- get_wfs_attributes("LIMITES_ADMINISTRATIVES_EXPRESS.LATEST:commune")
print(attrs)

plou_communes <- get_wfs(
  x = NULL,
  layer = "LIMITES_ADMINISTRATIVES_EXPRESS.LATEST:commune",
  query = "nom_officiel ILIKE 'PLOU%'"
)
plot(st_geometry(plou_communes))
```

```
# Multiple Attribute-only query (no spatial filter)
plou_inf_2000 <- get_wfs(
  x = NULL,
  layer = "LIMITES_ADMINISTRATIVES_EXPRESS.LATEST:commune",
  query = "nom_officiel ILIKE 'PLOU%' AND population < 2000"
)
plot(st_geometry(plou_communes))
plot(st_geometry(plou_inf_2000), col = "firebrick", add = TRUE)

# Spatial predicate usage

layer <- "BDCARTO_V5:rond_point"

bbox_feat <- get_wfs(commune, layer, predicate = bbox())
plot(st_geometry(bbox_feat), col = "red")
plot(st_geometry(commune), add = TRUE)

intersects_feat <- get_wfs(commune, layer, predicate = intersects())
plot(st_geometry(intersects_feat), col = "red")
plot(st_geometry(commune), add = TRUE)

dwithin_feat <- get_wfs(commune, layer, predicate = dwithin(5, "kilometers"))
plot(st_geometry(dwithin_feat), col = "red")
plot(st_geometry(commune), add = TRUE)

## 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)
```

Arguments

`layer`	`character`; name of the WFS layer. Must correspond to a layer available on the IGN WFS service (see `get_layers_metadata()`).

Value

`character`vector with layer attributes

Examples

```
## Not run:

get_wfs_attributes("LIMITES_ADMINISTRATIVES_EXPRESS.LATEST:commune")


## 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)
```

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.

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 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)
```

reg_2025 *French Regions Table (2025)*

Description

Data for French regions from the INSEE file "Régions".

Usage

```
reg_2025
```

Format

A data frame with one row per region and the following columns:

REG (int) Region code (2 characters)

CHEFLIEU_REG (chr) Commune code of the regional capital (5 characters)

TNCC_REG (int) Name type indicator (1 character)

NCC_REG (chr) Official name in uppercase (200 characters)

NCCENR_REG (chr) Official name with proper typography (200 characters)

LIBELLE_REG (chr) Official name with article and proper typography (200 characters)

Source

https://www.insee.fr/fr/statistiques/fichier/8377162/cog_ensemble_2025_csv.zip

spatial_predicates *Spatial predicate constructors*

Description

These functions create spatial predicates used by `get_wfs()` to filter features based on their spatial relationship with a reference geometry.

Usage

```
intersects()

within()

disjoint()

contains()

touches()

crosses()

overlaps()

equals()

bbox()

dwithin(distance, units = "meters")

beyond(distance, units = "meters")

relate(pattern)
```

Arguments

`distance`	Numeric distance value (single value).
`units`	Distance units supported by the WFS server (e.g. `"meters"`, `"kilometers"`).
`pattern`	A 9-character DE-9IM pattern string.

Details

Predicates describe *how* geometries should be compared (e.g. intersection, containment, distance-based relations). Users should not construct predicates manually; instead, use the helper functions listed below.

- `bbox()`: Select features intersecting the bounding box of the reference geometry.

- `intersects()`: Select features whose geometry intersects the reference geometry.
- `disjoint()`: Select features whose geometry intersects the reference geometry.
- `contains()`: Select features that completely contain the reference geometry.
- `within()`: Select features completely within the reference geometry.
- `touches()`: Select features that touch the reference geometry at the boundary.
- `crosses()`: Select features that cross the reference geometry.
- `overlaps()`: Select features that partially overlap the reference geometry.
- `equals()`: Select features geometrically equal to the reference geometry.
- `dwithin(distance, units)`: Select features within a given distance of the reference geometry.
- `beyond(distance, units)`: Select features farther than a given distance from the reference geometry.
- `relate(pattern)`: Select features matching a DE-9IM spatial relationship pattern.

Value

A spatial predicate object (used internally by `get_wfs()`).

See Also

`get_wfs()`

Examples

```
intersects()
bbox()
dwithin(50, "meters")
beyond(100, "meters")
relate("T*F**F***")
```

Index