**Package ‘happign’**

May 7, 2024

**Title** R Interface to 'IGN' Web Services

**Version** 0.3.0

**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)


**BugReports** https://github.com/paul-carerton/happign/issues

**Depends** R (>= 3.3.0)

**Imports** archive, dplyr, jsonlite, httr2 (>= 1.0.0), methods, sf (>= 1.0-7), terra, xml2

**Suggests** covr, httpstest2, knitr, rmarkdown, testthat (>= 3.0.0), tmap

**VignetteBuilder** knitr

**Config/testthat/edition** 3

**LazyData** true

**NeedsCompilation** no

**Encoding** UTF-8

**RoxygenNote** 7.3.0

**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** 2024-05-07 07:40:02 UTC
Description

Check if a wms layer is queryable with GetFeatureInfo.

Usage

\texttt{are\_queryable(apikey)}

Arguments

apikey \hspace{1cm} API key from \texttt{get\_apikeys()} or directly from the \texttt{IGN website}

Value

character containing the name of the queryable layers

See Also

\texttt{get\_location\_info()}
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

get_apicarto_cadastre  Apicarto Cadastre

Description

Implementation of the cadastre module from the IGN’s apicarto

Usage

gcget_apicarto_cadastre(x,
    type = "parcellle",
    source = "PCI",
    section = list(NULL),
    numero = list(NULL),
    code_arr = list(NULL),
    code_abs = list(NULL),
    code_com = list(NULL),
    dTolerance = 0)
get_apicarto_cadastre

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"

source
Can be "BDP" for BD Parcellaire or "PCI" for Parcellaire express. See detail for more info.

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

code_com
A character of length 5 corresponding to the commune code. Only use with type = "division" or type = "feuille"

dTolerance
numeric; Complex shape cannot be handle by API; using dTolerance allow to simplify them. See ?sf::st_simplify

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)

# 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")

## from insee_code
border <- get_apicarto_cadastre("29158", type = "commune")
borders <- get_apicarto_cadastre(c("29158", "29165"), type = "commune")
# get cadastral parcels
## from shape
parcels <- get_apicarto_cadastre(penmarch, section = "AX")

## from insee code
parcels <- get_apicarto_cadastre("29158")

# Use parameter recycling
## get sections "AX" parcels from multiple insee_code
parcels <- get_apicarto_cadastre(c("29158", "29165"), section = "AX")

## get parcels numbered "0001", "0010" of section "AX" and "BR"
section <- c("AX", "BR")
numero <- rep(c("0001", "0010"), each = 2)
parcels <- get_apicarto_cadastre("29158", section = section, numero = numero)

## generalization with expand.grid
params <- expand.grid(code_insee = c("29158", "29165"),
                      section = c("AX", "BR"),
                      numero = c("0001", "0010"),
                      stringsAsFactors = FALSE)
parcels <- get_apicarto_cadastre(params$code_insee,
                                 section = params$section,
                                 numero = params$numero)

## End(Not run)
get_apicarto_gpu

## Not run:

```r
info_commune <- get_apicarto_codes_postaux("29760")

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

## End(Not run)

---

### Description

Apicarto module Geoportail de l’urbanisme

### Usage

```r
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**.
- **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
get_apicarto_gpu

- "prescription-surf", "prescription-lin", "prescription-pct": it's a constraint or a possibility indicated in an urban planning document (PLU, PLUi, ...)
- "info-surf", "info-lin", "info-pct": it'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

```r
## 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",

## etc.
```
get_apicarto_rpg

"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 = list(NULL),
dTolerance = 0)

Arguments

x Object of class sf. Needs to be located in France.
annee numeric between 2010 and 2021
code_cultu character corresponding to code culture, see detail.
dTolerance numeric; tolerance parameter. The value of dTolerance must be specified in meters, see detail.

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_simplify.

Value

list or object of class sf
get_apicarto_viticole

Examples

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

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

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

# avoid complex data by setting dTolerance
rpg <- get_apicarto_rpg(penmarch, 2020, dTolerance = 10)

# multiple years after 2014
rpg <- get_apicarto_rpg(x, 2020:2021, dTolerance = 10)

# years before and after 2014
# list is returned because attributs are different
rpg <- get_apicarto_rpg(x, c(2010, 2021), dTolerance = 10)

# filter by code_cultu
rpg <- get_apicarto_rpg(x, 2021, code_cultu = "MIE", dTolerance = 10)

# all "MIE" from 2020 and all "PPH" from 2021
rpg <- get_apicarto_rpg(x, 2020:2021, code_cultu = c("MIE", "PPH"), dTolerance = 10)

# vectorization : all "MIE" from 2020 and 2021
rpg <- get_apicarto_rpg(x, 2020:2021, code_cultu = "MIE", dTolerance = 10)

## End(Not run)
```

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

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

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

```r
## Not run:
library(sf)
penmarch <- read_sf(system.file("extdata/penmarch.shp", package = "happign"))
VSIG <- get_apicarto_viticole(penmarch)
## End(Not run)
```

---

**Description**

All API keys are manually extract 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()
```
**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**

```r
get_last_news()
```

**Value**

message or error

**Examples**

```r
## 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**

```r
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**
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"
**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.

**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**

```r
## 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)
```
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

```r
## 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)
**Description**

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

**Usage**

```r
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

```r
## 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**

**Description**

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

**Usage**

```r
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**

character vector with layer attributes

**Examples**

```r
## Not run:
get_wfs_attributes("administratif", "LIMITES_ADMINISTRATIVES_EXPRESS.LATEST:commune")

# Interactive session
get_wfs_attributes(interactive = TRUE)

## End(Not run)
```

---

**get_wms_raster**

**Description**

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

Usage

get_wms_raster(x,
    layer = "ORTHOIMAGERY.ORTHOPHOTOS",
    res = 10,
    crs = 2154,
    rgb = TRUE,
    filename = tempfile(fileext = ".tif"),
    verbose = TRUE,
    overwrite = FALSE,
    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"). If NULL, uses layer as the filename. The default format is ".tif", but all GDAL drivers are supported.
- `verbose`: boolean; if TRUE, message are added.
- `overwrite`: boolean; if TRUE, the existing raster will be overwritten.
- `interactive`: logical; if TRUE, an interactive menu prompts for apikey and layer.

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.
- `overwrite`: The function `get_wms_raster` first checks if `filename` already exists. If it does, the file is imported into R without downloading again, unless `overwrite` is set to TRUE.

Value

SpatRaster object from terra package.

See Also

get_layers_metadata()
Examples

```r
## 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)
```

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

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

**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.
- **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**

```r
## 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)
```
get_wmts

## End(Not run)
Index

* datasets
  - cog_2023, 3
  - are_queryable, 2
  - cog_2023, 3
    - get_apicarto_cadastre, 3
    - get_apicarto_codes_postaux, 5
    - get_apicarto_gpu, 6
    - get_apicarto_rpg, 8
    - get_apicarto_viticole, 9
    - get_apikeys, 10
    - get_apikeys(), 12, 20
    - get_last_news, 11
    - get_layers_metadata, 11
    - get_layers_metadata(), 16, 18, 20
    - get_location_info, 12
    - get_location_info(), 2
    - get_raw_lidar, 13
    - get_wfs, 15
    - get_wfs_attributes, 17
    - get_wms_raster, 17
    - get_wmts, 19

  - sf::st_crs(), 18, 20