Package ‘happign’

April 12, 2023

Title  R Interface to 'IGN' Web Services
Version  0.1.9
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 (>= 3.3.0)
Imports  archive, dplyr, geojsonsf, htr2, methods, sf (>= 1.0-7),
         terra, xml2
Suggests  covr, httptest2, knitr, rmarkdown, testthat (>= 3.0.0), tmap
VignetteBuilder  knitr
Config/testthat/edition  3
Encoding  UTF-8
LazyData  true
NeedsCompilation  no
RoxygenNote  7.2.3
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  2023-04-12 09:00:02 UTC
R topics documented:

are_queryable .......................... 2
cog_2022 ................................ 3
get_apicarto_cadastre ..................... 3
get_apicarto_codes_postaux ............... 5
get_apicarto_commune ..................... 6
get_apicarto_gpu .......................... 7
get_apicarto_rpg ........................... 9
get_apicarto_viticole ..................... 10
get_apikeys ............................... 11
get_last_news ............................. 12
get_layers_metadata ....................... 12
get_raw_lidar ............................. 13
get_wfs .................................. 14
get_wfs_attributes ......................... 16
get_wms_info ............................. 17
get_wms_raster ............................ 18

Index 21

---

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

**Description**  
Check if a wms layer is queryable with GetFeatureInfo

**Usage**  
`are_queryable(apikey)`

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

**Value**  
character containing the name of the queryable layers

**See Also**  
`get_wms_info()`
Description

A dataset containing Insee code and 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 from the IGN’s apicarto

Usage

get_apicarto_cadastre(x,
  type = "parcelle",
  source = "PCI",
  section = list(NULL),
  numero = list(NULL),
  code_arr = list(NULL),
  code_abs = list(NULL),
  code_com = list(NULL))

Arguments

x  It can be a shape, Insee codes or department codes:
  • Shape: must be an object of class sf or sfc.
  • Code Insee: must be a character of length 5
  • Code department: must be a character of length 2 or 3 (DOM-TOM)
  type  A character from "parcelle" , "commune" , "feuille" , "division" , "localisant"
get_apicarto_cadastre

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"

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"

```r
section <- c("AX", "BR")
numero <- rep(c("0001", "0010"), each = 2)
parcels <- get_apicarto_cadastre("29158", section = section, numero = numero)
```

## generalization with expand.grid

```r
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_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**

```r
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**

```r
## 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_commune**  
*Apicarto Commune*

**Description**
Implementation of the cadastre module of the IGN’s apicarto for commune borders

**Usage**

```r
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_commune* return an object of class `sf`

**Examples**

```r
## 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")
```
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.
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 :
get_apicarto_gpu

- "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": 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)
ru <- get_apicarto_gpu("93014", "municipality")
ru$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,
```

```r
```
### 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

```r
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 attributes 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)
```

---

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**

```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_apiarto_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

```r
get_apikeys()
```

Value

character

Examples

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

# All API keys
get_apikeys()
```
get_last_news  \hspace{1cm} \textit{Print latest news from geoservice website}

\begin{description}
\item[Description] This function is a wrapper around the RSS feed of the geoservice site to get the latest information.
\item[Usage]\begin{verbatim}
get_last_news()
\end{verbatim}
\item[Value] message or error
\item[Examples]\begin{verbatim}
## Not run:
get_last_news()
## End(Not run)
\end{verbatim}
\end{description}

get_layers_metadata  \hspace{1cm} \textit{Metadata for one couple of apikey and data_type}

\begin{description}
\item[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.
\item[Usage] \begin{verbatim}
get_layers_metadata(apikey, data_type)
\end{verbatim}
\item[Arguments] \begin{verbatim}
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.
\end{verbatim}
\end{description}
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

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)
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

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

# Create shape
shape <- st_polygon(list(matrix(c(8.852234, 42.55466, 8.852234, 42.57289, 8.860474, 42.57289, 8.860474, 42.55466, 8.852234, 42.55466), ncol = 2, byrow = TRUE)))

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

# Download data to current directory
get_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

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

spatial_filter = "bbox",
ecql_filter = NULL,
overwrite = FALSE,
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, "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")
spatial_filter Character corresponding to a 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 If TRUE, file is overwrite
interactive if set to TRUE, no need to specify apikey and layer_name, 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. Be aware that "dwithin" is broken and it doesn’t accept units properly. Only degrees can be used. To avoid this, I recommend to use compute a buffer and use "within" instead od "dwithin".

• ECQL query can be provided to ecql_filter. This allows direct query of the IGN’s WFS geoservers. If shape is set, then the ecql_filter comes in addition to the spatial_filter. More info for writing ECQL here

Value

get_wfs return an object of class sf

See Also

get_apikeys(), 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(shape = penmarch,
                  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 <- metadata_table[32,1]

# Downloading borders
borders <- get_wfs(penmarch, 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_bdvl_shape")
qtm(forest_area, fill = "libelle")

# using ECQL filters to query IGN server

# find all commune's name starting by "plou".
# First you need the name of the attribute to filter
names(borders) # In our case "nom_m" is what we need

attribute_name <- names(get_wfs(penmarch, apikey, layer_name))
plouBorders <- get_wfs(shape = NULL, # When shape is NULL, all France is query
                        apikey = "administratif",
                        layer_name = "LIMITES_ADMINISTRATIVES_EXPRESS.LATEST:commune",
                        ecql_filter = "nom_m LIKE 'PLOU%'")

# it's also possible to combine ecql_filter
plouBorders <- get_wfs(shape = NULL, # When shape is NULL, all France is query
                        apikey = "administratif",
                        layer_name = "LIMITES_ADMINISTRATIVES_EXPRESS.LATEST:commune",
                        ecql_filter = "nom_m LIKE 'PLOU%' AND population < 2000")

## End(Not run)
get_wms_info

Description

get_wms_info

Usage

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

get_wfs_attributes

Description

get_wfs_attributes

Usage

get_wfs_attributes(apikey = NULL, layer_name = NULL, interactive = FALSE)

Arguments

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
interactive if set to TRUE, no need to specify apikey and layer_name, you’ll be ask.

Value

character vector with layers 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 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"),
               )

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"

Value

character containing additional information from the layer

Examples

## Not run:
library(sf)

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

wms_info <- get_wms_info(penmarch, "ortho", "ORTHOIMAGERY.ORTHOPHOTOS")

# orthophoto is from summer 2021
wms_info

## End(Not run)
get_wms_raster

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

```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(shape = penmarch, interactive = TRUE)

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

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

# 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

# Plotting result
tm_shape(mnt)+
  tm_raster(legend.show = FALSE)+
tm_shape(penmarch)+
  tm_borders(col = "blue", lwd = 3)

## End(Not run)
```
Index

* datasets
  cog_2022, 3

are_queryable, 2

cog_2022, 3

get_apicarto_cadastre, 3
get_apicarto_codes_postaux, 5
get_apicarto_commune, 6
get_apicarto_gpu, 7
get_apicarto_rpg, 9
get_apicarto_viticole, 10
get_apikeys, 11
get_apikeys(), 13, 15, 19
get_last_news, 12
get_layers_metadata, 12
get_layers_metadata(), 15, 19
get_raw_lidar, 13
get_wfs, 14
get_wfs_attributes, 16
get_wms_info, 17
get_wms_info(), 2
get_wms_raster, 18

sf::st_crs(), 19