Package: sigugr (via r-universe)

October 14, 2026

Type Package

Title Workflow for Geographic Data

Version 1.0.0.9000

Description Streamlines geographic data transformation, storage and publication, simplifying data preparation and enhancing interoperability across formats and platforms.

License MIT + file LICENSE

URL https://josesamos.github.io/sigugr/, https://github.com/josesamos/sigugr

BugReports https://github.com/josesamos/sigugr/issues

Depends R (>= 2.10)

Imports clc, gdalUtilities, httr, jsonlite, rpostgis, sf, snakecase, terra

Suggests httptest2, knitr, mockery, purrr, rmarkdown, testthat (>= 3.0.0), withr

VignetteBuilder knitr

Config/testthat/edition 3

Encoding UTF-8

Language en-GB

LazyData true

RoxygenNote 7.3.2

Config/pak/sysreqs libabsl-dev cmake libgdal-dev gdal-bin libgeos-dev libicu-dev libxml2-dev libssl-dev libpq-dev libproj-dev libsqlite3-dev libudunits2-dev

Repository https://josesamos.r-universe.dev

Date/Publication 2024-12-23 13:38:19 UTC

RemoteUrl https://github.com/josesamos/sigugr

RemoteRef HEAD

RemoteSha 6797766142f93cd7b9b2f5e6ee06529f9865416f

Contents

aggregate_rasters . . . 2
clip_layer . . . 3
clip_multipoligon . . . 4
clip_raster . . . 5
compose_raster . . . 6
copy_styles . . . 6
generate_bbox . . . 8
geoserver . . . 9
get_layer_categories . . . 10
publish_bands . . . 11
publish_layer . . . 12
publish_layer_set . . . 13
publish_raster . . . 15
register_datastore_postgis . . . 16
store_bands . . . 18
store_layers . . . 19
store_raster . . . 20

Index **22**

`aggregate_rasters` *Aggregate Rasters in a Folder*

Description

Aggregates all raster files ('.tif' or '.jp2') in a specified folder by the given factor and saves the resulting files in an output folder.

Usage

```
aggregate_rasters(dir, out_dir, factor = 2)
```

Arguments

`dir`	A string specifying the input folder containing raster files.
`out_dir`	A string specifying the output folder where the aggregated rasters will be saved.
`factor`	An integer specifying the aggregation factor (default is 2).

Details

If the output folder does not exist, it creates it.

Value

A character vector with the paths to the processed raster files.

See Also

Other transform raster functions: compose_raster()

Examples

```
temp_dir <- tempdir()
input_dir <- system.file("extdata", "mdt", package = "sigugr")

result_files <- aggregate_rasters(input_dir, temp_dir, factor = 4)
```

---

clip_layer *Clip a Vector Layer with a Polygon*

Description

Clips a vector layer using a polygon layer. It handles CRS transformations automatically if necessary, ensuring the output is in the same CRS as the input polygon.

Usage

```
clip_layer(vector, polygon)
```

Arguments

vector	An 'sf' object representing the vector layer to be clipped.
polygon	An 'sf' object representing the polygon layer used for clipping.

Value

An 'sf' object containing the features of the input 'vector' that intersect with the 'polygon'. The output will be in the CRS of the 'polygon', and it will retain all attributes of the input 'vector'.

See Also

Other clip functions: clip_multipoligon(), clip_raster(), generate_bbox()

Examples

```
gpkg_path <- system.file("extdata", "clc.gpkg", package = "clc")

clc <- sf::st_read(gpkg_path, layer = "clc", quiet = TRUE)
lanjaron <- sf::st_read(gpkg_path, layer = "lanjaron", quiet = TRUE)

clc_clipped <- clip_layer(clc, lanjaron)
```

`clip_multipoligon` *Safe Clip a Multipolygon Vector Layer*

Description

Clips a 'MULTIPOLYGON' vector layer using a polygon layer, handling specific issues that might arise with geometries encoded incorrectly or containing unknown WKB types. It serves as a fallback when the 'clip_layer' function fails due to errors like 'ParseException: Unknown WKB type 12', which is associated with *MULTIPOLYGON* types.

Usage

```
clip_multipoligon(vector, polygon)
```

Arguments

`vector`	A 'sf' multipolygon vector layer to be clipped.
`polygon`	A 'sf' polygon layer used as the clipping geometry.

Details

The function ensures that the input layer is correctly encoded as 'MULTIPOLYGON' and uses GDAL utilities for re-encoding if necessary. The output is projected to the CRS of the clipping polygon.

This solution is inspired by a discussion on handling WKB type errors in R: <https://gis.stackexchange.com/questions/389814 st-centroid-geos-error-unknown-wkb-type-12>.

Value

A 'sf' vector layer with the clipped geometries.

See Also

Other clip functions: `clip_layer()`, `clip_raster()`, `generate_bbox()`

Examples

```
gpkg_path <- system.file("extdata", "clc.gpkg", package = "clc")

clc <- sf::st_read(gpkg_path, layer = "clc", quiet = TRUE)
lanjaron <- sf::st_read(gpkg_path, layer = "lanjaron", quiet = TRUE)

clc_clipped <- clip_multipoligon(clc, lanjaron)
```

clip_raster *Clip a raster based on a polygon*

Description

Clips a raster using a polygon, preserving the coordinate reference system (CRS) of the raster.

Usage

```
clip_raster(raster, polygon, keep_crs = TRUE)
```

Arguments

raster	A ‘terra‘ raster to be clipped.
polygon	A ‘sf‘ polygon layer used for clipping.
keep_crs	Logical. If ‘TRUE‘, retains the original CRS of the raster. If ‘FALSE‘, transforms the raster to the polygon CRS. Default is ‘TRUE‘.

Value

A ‘terra‘ raster clipped to the extent of the polygon.

See Also

Other clip functions: clip_layer(), clip_multipoligon(), generate_bbox()

Examples

```
source_gpkg <- system.file("extdata", "sigugr.gpkg", package = "sigugr")
p <-sf::st_read(source_gpkg, layer = 'lanjaron', quiet = TRUE)

source_tif <- system.file("extdata", "sat.tif", package = "sigugr")
r <- terra::rast(source_tif)

result <- clip_raster(r, p)
```

compose_raster *Compose a Raster Layer from Multiple Files*

Description

Combines multiple raster files into a single virtual raster layer (VRT). It accepts one or more folder names containing raster files and creates a virtual raster file. If no output file name is provided, a temporary file is used.

Usage

```
compose_raster(dir, out_file = NULL)
```

Arguments

dir	A string or vector of strings representing folder names containing raster files.
out_file	A string specifying the output file name (without extension). If 'NULL', a temporary file is used.

Value

A 'SpatRaster' object from the 'terra' package.

See Also

Other transform raster functions: aggregate_rasters()

Examples

```
input_dir <- system.file("extdata", "mdt", package = "sigugr")

r <- compose_raster(input_dir)
```

copy_styles *Copy Styles from a Source to a Destination*

Description

Copies layer styles from a source (GeoPackage or PostGIS database) to a destination (GeoPackage or PostGIS database). The source and destination can be specified flexibly, and the function supports copying styles to multiple layers in the destination.

Usage

```
copy_styles(
  from,
  from_layer = NULL,
  to,
  database = NULL,
  schema = "public",
  to_layers = NULL
)
```

Arguments

from	A data source for the input style. This can be: - A string representing the path to a GeoPackage file. - A 'DBI' database connection object to a PostGIS database, created using [RPostgres::dbConnect()].
from_layer	Character (optional). Name of the layer in the source to copy the style from. If not provided, the function will use the first layer in the source with a defined style.
to	A data destination for the output styles. This can be: - A string representing the path to a GeoPackage file. - A 'DBI' database connection object to a PostGIS database, created using [RPostgres::dbConnect()].
database	Character (optional). Name of the destination PostGIS database (required if the destination is a PostGIS connection object).
schema	Character. Schema in the destination PostGIS database where the styles will be applied. Default is "public".
to_layers	Character vector (optional). Names of the layers in the destination where the style will be applied. If not provided, the style will be applied to all layers in the destination.

Value

The updated 'layer_styles' table, returned invisibly.

See Also

Other style functions: `get_layer_categories()`

Examples

```
# Ex1:
source_gpkg <- system.file("extdata", "clc.gpkg", package = "clc")
layer_data <- sf::st_read(source_gpkg, layer = "clc", quiet = TRUE)

dest_gpkg <- tempfile(fileext = ".gpkg")
sf::st_write(layer_data, dest_gpkg, layer = "clc", quiet = TRUE)

copy_styles(from = source_gpkg, to = dest_gpkg)
```

```
## Not run:
# Ex2:
source_gpkg <- system.file("extdata", "clc.gpkg", package = "clc")
conn <- DBI::dbConnect(
  RPostgres::Postgres(),
  dbname = "mydb",
  host = "localhost",
  user = "user",
  password = "password"
)

copy_styles(
  from = source_gpkg,
  to = conn,
  database = "mydb",
  schema = "public",
  to_layers = c("layer1", "layer2"),
)

DBI::dbDisconnect(conn)

## End(Not run)
```

`generate_bbox` *Generate a Bounding Box as an sf Object*

Description

Takes an ‘sf‘ object or a ‘terra::SpatRaster‘ as input and returns a new ‘sf‘ object representing the bounding box (minimum bounding rectangle) of the input layer.

Usage

```
generate_bbox(layer)
```

Arguments

`layer`	An ‘sf‘ object or a ‘terra::SpatRaster‘ object.

Value

An ‘sf‘ object representing the bounding box of the input layer.

See Also

Other clip functions: `clip_layer()`, `clip_multipoligon()`, `clip_raster()`

Examples

```
# Example with a vector layer
source_gpkg <- system.file("extdata/sigugr.gpkg", package = "sigugr")
lanjaron <- sf::st_read(source_gpkg, layer = "lanjaron", quiet = TRUE)
bbox_vector <- generate_bbox(lanjaron)

# Example with a raster layer
raster_file <- system.file("extdata/sat.tif", package = "sigugr")
raster <- terra::rast(raster_file)
bbox_raster <- generate_bbox(raster)
```

---

geoserver *GeoServer Connection Object ('geoserver' S3 Class)*

Description

This S3 class represents a connection to a GeoServer instance. It stores the connection details, including the base URL, user credentials, and the default workspace.

Usage

```
geoserver(url, user, password, workspace)
```

Arguments

`url`	A character string specifying the base URL of the GeoServer instance (e.g., '"http://localhost:8080/geoserver"').
`user`	A character string representing the GeoServer username with the required permissions.
`password`	A character string representing the password for the specified user.
`workspace`	A character string specifying the default workspace to use in GeoServer operations.

Value

An object of class 'geoserver' or NULL if an error occurred.

See Also

Other publish to GeoServer: `publish_bands()`, `publish_layer()`, `publish_layer_set()`, `publish_raster()`, `register_datastore_postgis()`

Examples

```
## Not run:
gso <- geoserver(
  url = "http://localhost:8080/geoserver",
  user = "admin",
  password = "geoserver",
  workspace = "sigugr_test"
)

## End(Not run)
```

`get_layer_categories` *Get Layer Categories Based on Raster Values*

Description

Extracts the categories (IDs, descriptions, and colors) from the first style definition stored in a GeoPackage or PostGIS database. The extracted categories are filtered to include only those present in the raster values.

Usage

```
get_layer_categories(from, r_clc = NULL)
```

Arguments

`from`	A data origin. This can be: - A string representing the path to a GeoPackage file. - A ‘DBI‘ database connection object to a PostGIS database, created using [RPostgres::dbConnect()].
`r_clc`	A ‘terra‘ raster object containing the raster values to filter the categories. If NULL, returns all categories.

Details

The function retrieves the style definitions from the ‘layer_styles‘ table in the provided GeoPackage or PostGIS database. It filters the categories to include only those whose IDs match the unique values present in the raster.

It is useful for associating raster values with their corresponding descriptions and colors, typically for visualization or analysis tasks.

Value

A data frame containing the filtered categories with the following columns: - ‘id‘: The category ID (integer). - ‘description‘: The description of the category (character). - ‘color‘: The color associated with the category in hexadecimal format (character).

See Also

Other style functions: copy_styles()

Examples

```
gpkg_path <- system.file("extdata", "clc.gpkg", package = "clc")

categories <- get_layer_categories(from = gpkg_path)
```

---

publish_bands *Publish Bands of a Raster to GeoServer*

Description

Publishes bands of a multi-band GeoTIFF raster file as separate coverages in a specified workspace on a GeoServer instance.

Usage

```
publish_bands(gso, raster, prefix, postfix, bands)

## S3 method for class 'geoserver'
publish_bands(gso, raster, prefix = NULL, postfix = NULL, bands = NULL)
```

Arguments

gso	An object of class ‘geoserver‘ containing GeoServer connection details.
raster	A character string specifying the file path to the GeoTIFF raster file to be uploaded.
prefix	A string to prepend to each layer name. Default is ‘NULL‘.
postfix	A string to append to each layer name. Default is ‘NULL‘.
bands	A named integer vector, index of the bands to publish with layer names. If it is ‘NULL‘, which is the default value, all bands are published using the band name as the layer name. If unnamed indices are provided, the band name is also used as the layer name.

Value

An integer:

- 0 if the operation was successful or if the layer already exists.
- 1 if an error occurred.

See Also

Other publish to GeoServer: geoserver(), publish_layer(), publish_layer_set(), publish_raster(), register_datastore_postgis()

Examples

```
## Not run:
gso <- geoserver(
  url = "http://localhost:8080/geoserver",
  user = "admin",
  password = "geoserver",
  workspace = "sigugr_test"
)

source_tif <- system.file("extdata/sat.tif", package = "sigugr")

gso |>
  publish_bands(source_tif)

## End(Not run)
```

`publish_layer` *Publish a Vector Layer to GeoServer*

Description

Publishes a vector layer to GeoServer. The layer source must have previously been defined as a GeoSever datastore.

Usage

```
publish_layer(gso, layer, title)

## S3 method for class 'geoserver'
publish_layer(gso, layer, title = NULL)
```

Arguments

`gso`	An object of class ‘geoserver‘ containing GeoServer connection details.
`layer`	A string, the name of the layer to publish.
`title`	A string, an optional title for the layer. Defaults to the layer name if not provided.

Details

Prints an appropriate message indicating success or failure.

Value

An integer:

- `0` if the operation was successful or if the layer already exists.
- `1` if an error occurred.

See Also

Other publish to GeoServer: geoserver(), publish_bands(), publish_layer_set(), publish_raster(), register_datastore_postgis()

Examples

```
## Not run:
gso <- geoserver(
  url = "http://localhost:8080/geoserver",
  user = "admin",
  password = "geoserver",
  workspace = "sigugr_test"
)

gso <- gso |>
  register_datastore_postgis(
    "sigugr-postgis",
    db_name = 'sigugr_example',
    host = 'localhost',
    port = 5432,
    db_user = 'user',
    db_password = 'password',
    schema = "public"
  )

gso |>
  publish_layer(layer = 'sigugr_layer')

## End(Not run)
```

---

publish_layer_set *Publish a Vector Layer Set to GeoServer*

Description

Publishes a vector layer set to GeoServer. The layer source must have previously been defined as a GeoSever datastore.

Usage

```
publish_layer_set(gso, source, layers)

## S3 method for class 'geoserver'
publish_layer_set(gso, source, layers = NULL)
```

Arguments

gso	An object of class ‘geoserver‘ containing GeoServer connection details.
source	A valid connection to a PostGIS database (‘RPostgres‘ connection object).
layers	An optional character vector of layer names to check and publish. If ‘NULL‘ (default), all vector geometry layers in the source will be published.

Details

Iterates over a set of layers in a source database, checking whether each layer contains vector geometry. If the layer meets the criteria, it is published. If the ‘layers‘ parameter is ‘NULL‘, the function will publish all layers with vector geometry in the source.

Prints an appropriate messages indicating success or failure.

Value

An integer:

- `0` if the operation was successful for all layers.
- `1` if an error occurred.

See Also

Other publish to GeoServer: `geoserver()`, `publish_bands()`, `publish_layer()`, `publish_raster()`, `register_datastore_postgis()`

Examples

```
## Not run:
gso <- geoserver(
  url = "http://localhost:8080/geoserver",
  user = "admin",
  password = "geoserver",
  workspace = "sigugr_test"
)

gso <- gso |>
  register_datastore_postgis(
    "sigugr-postgis",
    db_name = 'sigugr_example',
    host = 'localhost',
    port = 5432,
    db_user = 'user',
    db_password = 'password',
    schema = "public"
  )

source <- RPostgres::dbConnect(
  RPostgres::Postgres(),
  dbname = 'mydb',
  host = 'localhost',
```

```
  port = '5432',
  user = 'user',
  password = 'password'
)

gso |>
  publish_layer_set(source)

## End(Not run)
```

`publish_raster` *Publish a Raster to GeoServer*

Description

Publishes a GeoTIFF raster file to a workspace and data store on a GeoServer instance.

Usage

```
publish_raster(gso, raster, layer)

## S3 method for class 'geoserver'
publish_raster(gso, raster, layer = NULL)
```

Arguments

`gso`	An object of class ‘geoserver‘ containing GeoServer connection details.
`raster`	A character string specifying the file path to the GeoTIFF raster file to be uploaded.
`layer`	A string, the name of the layer to publish. If it is ‘NULL‘, which is the default value, the layer name is derived from the filename.

Value

An integer:

- `0` if the operation was successful or if the layer already exists.
- `1` if an error occurred.

See Also

Other publish to GeoServer: `geoserver()`, `publish_bands()`, `publish_layer()`, `publish_layer_set()`, `register_datastore_postgis()`

Examples

```
## Not run:
gso <- geoserver(
  url = "http://localhost:8080/geoserver",
  user = "admin",
  password = "geoserver",
  workspace = "sigugr_test"
)

source_tif <- system.file("extdata/sat.tif", package = "sigugr")

gso |>
  publish_raster(source_tif, "sat-tiff")

## End(Not run)
```

`register_datastore_postgis` *Register a PostGIS Database as a DataStore in GeoServer*

Description

Registers a PostGIS database as a ‘datastore‘ in a specified GeoServer workspace.

Usage

```
register_datastore_postgis(
  gso,
  datastore,
  db_name,
  host,
  port,
  db_user,
  db_password,
  schema
)

## S3 method for class 'geoserver'
register_datastore_postgis(
  gso,
  datastore,
  db_name,
  host,
  port = 5432,
  db_user,
  db_password,
  schema = "public"
)
```

Arguments

gso	An object of class ‘geoserver‘ containing GeoServer connection details.
datastore	A character string. The name of the datastore to be created.
db_name	A character string. The name of the PostGIS database.
host	A character string. The database host.
port	An integer. The database port (default: 5432).
db_user	A character string. The database username.
db_password	A character string. The database password.
schema	A character string. The database schema (default: "public").

Details

If the ‘datastore‘ has already been registered previously, there is no need to specify the database connection. For subsequent operations, that ‘datastore‘ will be used.

In any case, prints an appropriate message.

Value

An object of class ‘geoserver‘ or NULL if an error occurred.

See Also

Other publish to GeoServer: geoserver(), publish_bands(), publish_layer(), publish_layer_set(), publish_raster()

Examples

```
## Not run:
gso <- geoserver(
  url = "http://localhost:8080/geoserver",
  user = "admin",
  password = "geoserver",
  workspace = "sigugr_test"
)

gso <- gso |>
  register_datastore_postgis(
    "sigugr-postgis",
    db_name = 'sigugr_example',
    host = 'localhost',
    port = 5432,
    db_user = 'user',
    db_password = 'password',
    schema = "public"
  )

## End(Not run)
```

store_bands *Store Raster Bands to PostGIS*

Description

Stores each band of a raster to a specified schema in a PostGIS database. Each band is written as a separate table in the database.

Usage

```
store_bands(
  raster,
  conn,
  schema = "public",
  prefix = NULL,
  postfix = NULL,
  bands = NULL
)
```

Arguments

raster	A character string specifying the file path to the GeoTIFF file containing the raster bands to be stored.
conn	A database connection object to a PostGIS database (e.g., from ‘RPostgres::dbConnect‘).
schema	A string specifying the schema in the PostGIS database where the raster layers will be stored. Default is ‘"public"‘.
prefix	A string to prepend to each layer name. Default is ‘NULL‘.
postfix	A string to append to each layer name. Default is ‘NULL‘.
bands	A named integer vector, index of the bands to store with layer names. If it is ‘NULL‘, which is the default value, all bands are stored using the band name as the layer name. If unnamed indices are provided, the band name is also used as the layer name.

Details

Transforms the table name according to the Snake Case convention.

Value

Invisibly returns a character vector of the names of the tables written to PostGIS.

See Also

Other write to PostGIS: store_layers(), store_raster()

Examples

```
## Not run:
conn <- DBI::dbConnect(
  RPostgres::Postgres(),
  dbname = "mydb",
  host = "localhost",
  user = "user",
  password = "password"
)

sr <- terra::rast(nrows = 10, ncols = 10, nlyrs = 3, vals = runif(300))
sr_file <- tempfile(fileext = ".tif")
terra::writeRaster(sr, sr_file, filetype = "GTiff", overwrite = TRUE)

tables <- store_bands(sr_file, conn, schema = "geodata", prefix = "example_", postfix = "_raster")

DBI::dbDisconnect(conn)

## End(Not run)
```

---

store_layers *Write GeoPackage Layers with Geometry to PostGIS*

Description

Transfers vector layers with valid geometries from a GeoPackage file to a specified PostGIS database schema. Optionally allows setting a custom geometry column name, adding prefixes or postfixes to the table names, and renaming the layer fields to follow the Snake Case convention.

Usage

```
store_layers(
  gpkg,
  conn,
  schema = "public",
  prefix = NULL,
  postfix = NULL,
  layers = NULL,
  geom_colum = "geom",
  snake_case_fields = TRUE
)
```

Arguments

gpkg	A string, the path to the GeoPackage file.
conn	A PostGIS database connection object created with [RPostgres::dbConnect()].

schema	A string, the schema in PostGIS where layers will be stored. Default is ‘"public"‘.
prefix	A string, an optional prefix to add to the table names in PostGIS. Default is ‘NULL‘.
postfix	A string, an optional postfix to add to the table names in PostGIS. Default is ‘NULL‘.
layers	A string vector, the name of the layers to transfer. If NULL, all vector layers are transferred.
geom_colum	A string, the name of the geometry column to set. Default is ‘"geom"‘.
snake_case_fields	A logical, whether to convert field names to Snake Case. Default is ‘TRUE‘.

Value

Invisibly returns a character vector of the names of the tables written to PostGIS.

See Also

Other write to PostGIS: store_bands(), store_raster()

Examples

```
## Not run:
source_gpkg <- system.file("extdata", "sigugr.gpkg", package = "sigugr")

conn <- DBI::dbConnect(
  RPostgres::Postgres(),
  dbname = "mydb",
  host = "localhost",
  user = "user",
  password = "password"
)

store_layers(
  source_gpkg, conn, prefix = "pre_", postfix = "_post"
)

DBI::dbDisconnect(conn)

## End(Not run)
```

---

store_raster *Store Raster to PostGIS*

Description

Stores all bands of a raster to a specified schema in a PostGIS database. All bands are written in the same table in the database.

Usage

```
store_raster(raster, conn, schema = "public", table_name = NULL)
```

Arguments

raster	A character string specifying the file path to the GeoTIFF raster file to be stored.
conn	A database connection object to a PostGIS database (e.g., from 'RPostgres::dbConnect').
schema	A string specifying the schema in the PostGIS database where the raster layers will be stored. Default is '"public"'.
table_name	A string, table name. If it is 'NULL', which is the default value, the layer name is derived from the filename.

Details

Transforms the table name according to the Snake Case convention.

Value

Invisibly returns a character vector of the names of the tables written to PostGIS.

See Also

Other write to PostGIS: store_bands(), store_layers()

Examples

```
## Not run:
source_tif <- system.file("extdata", "mdt.tif", package = "clc")

conn <- DBI::dbConnect(
  RPostgres::Postgres(),
  dbname = "mydb",
  host = "localhost",
  user = "user",
  password = "password"
)

tables <- store_raster(source_tif, conn, table_name = "mdt")

DBI::dbDisconnect(conn)

## End(Not run)
```

Index

∗ **clip functions**
  clip_layer, 3
  clip_multipoligon, 4
  clip_raster, 5
  generate_bbox, 8
∗ **publish to GeoServer**
  geoserver, 9
  publish_bands, 11
  publish_layer, 12
  publish_layer_set, 13
  publish_raster, 15
  register_datastore_postgis, 16
∗ **style functions**
  copy_styles, 6
  get_layer_categories, 10
∗ **transform raster functions**
  aggregate_rasters, 2
  compose_raster, 6
∗ **write to PostGIS**
  store_bands, 18
  store_layers, 19
  store_raster, 20

aggregate_rasters, 2, *6*

clip_layer, 3, *4, 5*, *8*
clip_multipoligon, *3*, 4, *5*, *8*
clip_raster, *3, 4*, 5, *8*
compose_raster, *3*, 6
copy_styles, 6, *11*

generate_bbox, *3–5*, 8
geoserver, 9, *11*, *13–15*, *17*
get_layer_categories, *7*, 10

publish_bands, *9*, 11, *13–15*, *17*
publish_layer, *9*, *11*, 12, *14, 15*, *17*
publish_layer_set, *9*, *11*, 13, *13*, *15*, *17*
publish_raster, *9*, *11*, *13, 14*, 15, *17*

register_datastore_postgis, *9*, *11*, *13–15*, 16

store_bands, 18, *20, 21*
store_layers, *18*, 19, *21*
store_raster, *18*, 20, *20*