Package: rsi (via r-universe)

June 28, 2024

Title Efficiently Retrieve and Process Satellite Imagery

Version 0.2.1

Description Downloads spatial data from spatiotemporal asset catalogs ('STAC'), computes standard spectral indices from the Awesome Spectral Indices project (Montero et al. (2023) <doi:10.1038/s41597-023-02096-0>) against raster data, and glues the outputs together into predictor bricks. Methods focus on interoperability with the broader spatial ecosystem; function arguments and outputs use classes from 'sf' and 'terra', and data downloading functions support complex 'CQL2' queries using 'rstac'.

License Apache License (>= 2)

Depends R (>= 4.0)

Imports future.apply, glue, httr, jsonlite, lifecycle, proceduralnames, rlang, rstac, sf, terra, tibble

Suggests curl, knitr, progressr, rmarkdown, testthat (>= 3.0.0), withr

Config/testthat/edition 3

Config/testthat/parallel true

Encoding UTF-8

LazyData true

RoxygenNote 7.3.1

URL https://github.com/Permian-Global-Research/rsi, https://permian-global-research.github.io/rsi/

BugReports https://github.com/Permian-Global-Research/rsi/issues

VignetteBuilder knitr

NeedsCompilation no

Author Michael Mahoney [aut, cre] (<https://orcid.org/0000-0003-2402-304X>), Permian Global [cph, fnd]

Maintainer Michael Mahoney <mike.mahoney.218@gmail.com>

Repository CRAN

Date/Publication 2024-06-27 19:40:02 UTC

Contents

`alos_palsar_band_mapping`
ALOS PALSAR band mapping

Description

This object is a named list of character vectors, with names corresponding to Landsat band names and values corresponding to band names in `spectral_indices`.

Usage

```
alos_palsar_band_mapping
```

Format

An object of class `list` of length 1.

Details

Band mapping objects:

These objects are semi-standardized sets of metadata which provide all the necessary information for downloading data from a given STAC server. The object itself is list of character vectors, whose names represent asset names on a given STAC server and whose values represent the corresponding standardized band name from the Awesome Spectral Indices project. In addition to this data, these vectors usually have some of (but not necessarily all of) the following attributes:

- `stac_source`: The URL for the STAC server this metadata corresponds to.
- `collection_name`: The default STAC collection for this data source.
- `download_function`: The function to be used to download assets from the STAC server.
- `mask_band`: The name of the asset on this server to be used for masking images.
- `mask_function`: The function to be used to mask images downloaded from this server.

---

alos_palsar_mask_function

Create an ALOS PALSAR mask raster from the mask band

Description

Create an ALOS PALSAR mask raster from the mask band

Usage

```
alos_palsar_mask_function(raster, include = c("land", "water", "both"))
```

Arguments

`raster`	The mask band of an ALOS PALSAR image
`include`	Include pixels that represent land, water, or both? Passing `c("land", "water")` is identical to passing `"both"`.

Value

A boolean raster to be used to mask an ALOS PALSAR image

Examples

```
aoi <- sf::st_point(c(-74.912131, 44.080410))
aoi <- sf::st_set_crs(sf::st_sfc(aoi), 4326)
aoi <- sf::st_buffer(sf::st_transform(aoi, 5070), 100)

palsar_image <- get_alos_palsar_imagery(
  aoi,
  start_date = "2021-01-01",
  end_date = "2021-12-31",
```

```
  mask_function = alos_palsar_mask_function
)
```

`calculate_indices` *Calculate indices from the bands of a raster*

Description

This function computes any number of indices from an input raster via `terra::predict()`. By default, this function is designed to work with subsets of `spectral_indices()`, but it will work with any data frame with a `formula`, `bands`, and `short_name` column.

Usage

```
calculate_indices(
  raster,
  indices,
  output_filename,
  ...,
  cores = 1L,
  wopt = list(),
  overwrite = FALSE,
  extra_objects = list(),
  names_suffix = NULL
)
```

Arguments

`raster`	The raster (either as a SpatRaster or object readable by `terra::rast()`) to compute indices from.
`indices`	A data frame of indices to compute. The intent is for this function to work with subsets of spectral_indices, but any data frame with columns `formula` (containing a string representation of the equation used to calculate the index), `bands` (a list column containing character vectors of the necessary bands) and `short_name` (which will be used as the band name) will work.
`output_filename`	The filename to write the computed metrics to.
`...`	These dots are for future extensions and must be empty.
`cores`	positive integer. If `cores` > 1, a 'parallel' package cluster with that many cores is created and used
`wopt`	list with named options for writing files as in `writeRaster`
`overwrite`	logical. If `TRUE`, `filename` is overwritten

`extra_objects` A named list of additional objects to pass to the minimal environment that formulas are executed in. For instance, if you need to use the `pmax` function in order to calculate an index, you can make it available in the environment by setting `extra_objects = list("pmax" = pmax)`. Providing extra functionality is inherently less safe than the default minimal environment, and as such always emits a warning, which you can suppress with `suppressWarnings()`.

`names_suffix` If not `NULL`, will be used (with `paste()`) to add a suffix to each of the band names returned.

Value

`output_filename`, unchanged.

Security

Note that this function is running code from the `formula` column of the spectral indices data frame, which is derived from a JSON file downloaded off the internet. It's not impossible that an attacker could take advantage of this to run arbitrary code on your computer. To mitigate this, indices are calculated in a minimal environment that contains very few functions or symbols (preventing an attacker from accessing, for example, `system()`).

Still, it's good practice to inspect your `formula` column to make sure there's nothing nasty hiding in any of the formulas you're going to run. Additionally, consider using pre-saved indices tables or `spectral_indices(download_indices = FALSE)` if using this in an unsupervised workload.

Examples

```
our_raster <- system.file("rasters/example_sentinel1.tif", package = "rsi")
calculate_indices(
  our_raster,
  filter_bands(bands = names(terra::rast(our_raster))),
  tempfile(fileext = ".tif"),
  names_suffix = "sentinel1"
)

# Formulas aren't able to access most R functions or operators:
example_indices <- filter_platforms(platforms = "Sentinel-1 (Dual Polarisation VV-VH)")[1, ]
example_indices$formula <- 'system("echo something bad")'
try(
  calculate_indices(
    system.file("rasters/example_sentinel1.tif", package = "rsi"),
    example_indices,
    tempfile(fileext = ".tif")
  )
)
```

dem_band_mapping *Landsat band mapping*

Description

This object is structured slightly differently from other band mapping objects; it is a list of named lists, whose names correspond to DEM collections available within a given STAC catalog. Those named lists are then more standard band mapping objects, containing character vectors with names corresponding to asset names and values equal to `elevation`.

Usage

```
dem_band_mapping
```

Format

An object of class `list` of length 1.

Details

Band mapping objects:

These objects are semi-standardized sets of metadata which provide all the necessary information for downloading data from a given STAC server. The object itself is list of character vectors, whose names represent asset names on a given STAC server and whose values represent the corresponding standardized band name from the Awesome Spectral Indices project. In addition to this data, these vectors usually have some of (but not necessarily all of) the following attributes:

- `stac_source`: The URL for the STAC server this metadata corresponds to.
- `collection_name`: The default STAC collection for this data source.
- `download_function`: The function to be used to download assets from the STAC server.
- `mask_band`: The name of the asset on this server to be used for masking images.
- `mask_function`: The function to be used to mask images downloaded from this server.

filter_platforms *Filter indices based on (relatively) complicated fields*

Description

Filter indices based on (relatively) complicated fields

Usage

```
filter_platforms(
  indices = spectral_indices(),
 platforms = unique(unlist(spectral_indices(download_indices = FALSE, update_cache =
    FALSE)$platforms)),
  operand = c("all", "any")
)

filter_bands(
  indices = spectral_indices(),
 bands = unique(unlist(spectral_indices(download_indices = FALSE, update_cache =
    FALSE)$bands)),
  operand = c("all", "any"),
  type = c("filter", "search")
)
```

Arguments

indices	The data frame to filter. Must contain the relevant column.
platforms, bands	Names of the instruments (for platforms) or spectra (for bands) indices must contain.
operand	A function defining how to apply this filter. For instance, operand = all means that the index must contain all the platforms or bands provided, while operand = any means that the index must contain at least one of the platforms or bands provided.
type	What type of query is this? If filter, then indices are returned if all/any the bands they use (depending on operand) are in bands. If search, then indices are returned if all/any of bands are in the bands they use.

Value

A filtered version of indices.

Examples

```
filter_platforms(platforms = "Sentinel-2")
filter_platforms(platforms = c("Landsat-OLI", "Sentinel-2"))
filter_bands(bands = c("R", "N"), operand = any)
```

---

get_stac_data *Retrieve raster data from STAC endpoints*

Description

These functions retrieve raster data from STAC endpoints and optionally create composite data sets from multiple files. `get_stac_data()` is a generic function which should be able to download raster data from a variety of data sources, while the other helper functions have useful defaults for downloading common data sets from standard STAC sources.

Usage

```
get_stac_data(
  aoi,
  start_date,
  end_date,
  pixel_x_size = NULL,
  pixel_y_size = NULL,
  asset_names,
  stac_source,
  collection,
  ...,
  query_function = rsi_query_api,
  download_function = rsi_download_rasters,
  sign_function = NULL,
  rescale_bands = TRUE,
  item_filter_function = NULL,
  mask_band = NULL,
  mask_function = NULL,
  output_filename = paste0(proceduralnames::make_english_names(1), ".tif"),
  composite_function = c("merge", "median", "mean", "sum", "min", "max"),
  limit = 999,
  gdalwarp_options = c("-r", "bilinear", "-multi", "-overwrite", "-co",
    "COMPRESS=DEFLATE", "-co", "PREDICTOR=2", "-co", "NUM_THREADS=ALL_CPUS"),
 gdal_config_options = c(VSI_CACHE = "TRUE", GDAL_CACHEMAX = "30%", VSI_CACHE_SIZE =
  "10000000", GDAL_HTTP_MULTIPLEX = "YES", GDAL_INGESTED_BYTES_AT_OPEN = "32000",
    GDAL_DISABLE_READDIR_ON_OPEN = "EMPTY_DIR", GDAL_HTTP_VERSION = "2",
    GDAL_HTTP_MERGE_CONSECUTIVE_RANGES = "YES", GDAL_NUM_THREADS = "ALL_CPUS",
   GDAL_HTTP_USERAGENT = "rsi (https://permian-global-research.github.io/rsi/)")
)

get_sentinel1_imagery(
  aoi,
  start_date,
  end_date,
  ...,
  pixel_x_size = 10,
  pixel_y_size = 10,
  asset_names = rsi::sentinel1_band_mapping$planetary_computer_v1,
  stac_source = attr(asset_names, "stac_source"),
  collection = attr(asset_names, "collection_name"),
  query_function = attr(asset_names, "query_function"),
```

```
  download_function = attr(asset_names, "download_function"),
  sign_function = attr(asset_names, "sign_function"),
  rescale_bands = FALSE,
  item_filter_function = NULL,
  mask_band = NULL,
  mask_function = NULL,
  output_filename = paste0(proceduralnames::make_english_names(1), ".tif"),
  composite_function = "median",
  limit = 999,
  gdalwarp_options = c("-r", "bilinear", "-multi", "-overwrite", "-co",
    "COMPRESS=DEFLATE", "-co", "PREDICTOR=2", "-co", "NUM_THREADS=ALL_CPUS"),
 gdal_config_options = c(VSI_CACHE = "TRUE", GDAL_CACHEMAX = "30%", VSI_CACHE_SIZE =
  "10000000", GDAL_HTTP_MULTIPLEX = "YES", GDAL_INGESTED_BYTES_AT_OPEN = "32000",
    GDAL_DISABLE_READDIR_ON_OPEN = "EMPTY_DIR", GDAL_HTTP_VERSION = "2",
    GDAL_HTTP_MERGE_CONSECUTIVE_RANGES = "YES", GDAL_NUM_THREADS = "ALL_CPUS")
)

get_sentinel2_imagery(
  aoi,
  start_date,
  end_date,
  ...,
  pixel_x_size = 10,
  pixel_y_size = 10,
  asset_names = rsi::sentinel2_band_mapping$planetary_computer_v1,
  stac_source = attr(asset_names, "stac_source"),
  collection = attr(asset_names, "collection_name"),
  query_function = attr(asset_names, "query_function"),
  download_function = attr(asset_names, "download_function"),
  sign_function = attr(asset_names, "sign_function"),
  rescale_bands = FALSE,
  item_filter_function = NULL,
  mask_band = attr(asset_names, "mask_band"),
  mask_function = attr(asset_names, "mask_function"),
  output_filename = paste0(proceduralnames::make_english_names(1), ".tif"),
  composite_function = "median",
  limit = 999,
  gdalwarp_options = c("-r", "bilinear", "-multi", "-overwrite", "-co",
    "COMPRESS=DEFLATE", "-co", "PREDICTOR=2", "-co", "NUM_THREADS=ALL_CPUS"),
 gdal_config_options = c(VSI_CACHE = "TRUE", GDAL_CACHEMAX = "30%", VSI_CACHE_SIZE =
  "10000000", GDAL_HTTP_MULTIPLEX = "YES", GDAL_INGESTED_BYTES_AT_OPEN = "32000",
    GDAL_DISABLE_READDIR_ON_OPEN = "EMPTY_DIR", GDAL_HTTP_VERSION = "2",
    GDAL_HTTP_MERGE_CONSECUTIVE_RANGES = "YES", GDAL_NUM_THREADS = "ALL_CPUS")
)

get_landsat_imagery(
  aoi,
  start_date,
```

```
  end_date,
  ...,
  platforms = c("landsat-9", "landsat-8"),
  pixel_x_size = 30,
  pixel_y_size = 30,
  asset_names = rsi::landsat_band_mapping$planetary_computer_v1,
  stac_source = attr(asset_names, "stac_source"),
  collection = attr(asset_names, "collection_name"),
  query_function = attr(asset_names, "query_function"),
  download_function = attr(asset_names, "download_function"),
  sign_function = attr(asset_names, "sign_function"),
  rescale_bands = TRUE,
  item_filter_function = landsat_platform_filter,
  mask_band = attr(asset_names, "mask_band"),
  mask_function = attr(asset_names, "mask_function"),
  output_filename = paste0(proceduralnames::make_english_names(1), ".tif"),
  composite_function = "median",
  limit = 999,
  gdalwarp_options = c("-r", "bilinear", "-multi", "-overwrite", "-co",
    "COMPRESS=DEFLATE", "-co", "PREDICTOR=2", "-co", "NUM_THREADS=ALL_CPUS"),
 gdal_config_options = c(VSI_CACHE = "TRUE", GDAL_CACHEMAX = "30%", VSI_CACHE_SIZE =
  "10000000", GDAL_HTTP_MULTIPLEX = "YES", GDAL_INGESTED_BYTES_AT_OPEN = "32000",
    GDAL_DISABLE_READDIR_ON_OPEN = "EMPTY_DIR", GDAL_HTTP_VERSION = "2",
    GDAL_HTTP_MERGE_CONSECUTIVE_RANGES = "YES", GDAL_NUM_THREADS = "ALL_CPUS")
)

get_naip_imagery(
  aoi,
  start_date,
  end_date,
  ...,
  pixel_x_size = 1,
  pixel_y_size = 1,
  asset_names = "image",
  stac_source = "https://planetarycomputer.microsoft.com/api/stac/v1",
  collection = "naip",
  query_function = rsi_query_api,
  download_function = rsi_download_rasters,
  sign_function = sign_planetary_computer,
  rescale_bands = FALSE,
  output_filename = paste0(proceduralnames::make_english_names(1), ".tif"),
  composite_function = "merge",
  limit = 999,
  gdalwarp_options = c("-r", "bilinear", "-multi", "-overwrite", "-co",
    "COMPRESS=DEFLATE", "-co", "PREDICTOR=2", "-co", "NUM_THREADS=ALL_CPUS"),
 gdal_config_options = c(VSI_CACHE = "TRUE", GDAL_CACHEMAX = "30%", VSI_CACHE_SIZE =
  "10000000", GDAL_HTTP_MULTIPLEX = "YES", GDAL_INGESTED_BYTES_AT_OPEN = "32000",
    GDAL_DISABLE_READDIR_ON_OPEN = "EMPTY_DIR", GDAL_HTTP_VERSION = "2",
```

```
    GDAL_HTTP_MERGE_CONSECUTIVE_RANGES = "YES", GDAL_NUM_THREADS = "ALL_CPUS")
)

get_alos_palsar_imagery(
  aoi,
  start_date,
  end_date,
  ...,
  pixel_x_size = 25,
  pixel_y_size = 25,
  asset_names = rsi::alos_palsar_band_mapping$planetary_computer_v1,
  stac_source = attr(asset_names, "stac_source"),
  collection = attr(asset_names, "collection_name"),
  query_function = attr(asset_names, "query_function"),
  download_function = attr(asset_names, "download_function"),
  sign_function = attr(asset_names, "sign_function"),
  rescale_bands = FALSE,
  item_filter_function = NULL,
  mask_band = attr(asset_names, "mask_band"),
  mask_function = attr(asset_names, "mask_function"),
  output_filename = paste0(proceduralnames::make_english_names(1), ".tif"),
  composite_function = "median",
  limit = 999,
  gdalwarp_options = c("-r", "bilinear", "-multi", "-overwrite", "-co",
    "COMPRESS=DEFLATE", "-co", "PREDICTOR=2", "-co", "NUM_THREADS=ALL_CPUS"),
 gdal_config_options = c(VSI_CACHE = "TRUE", GDAL_CACHEMAX = "30%", VSI_CACHE_SIZE =
  "10000000", GDAL_HTTP_MULTIPLEX = "YES", GDAL_INGESTED_BYTES_AT_OPEN = "32000",
    GDAL_DISABLE_READDIR_ON_OPEN = "EMPTY_DIR", GDAL_HTTP_VERSION = "2",
    GDAL_HTTP_MERGE_CONSECUTIVE_RANGES = "YES", GDAL_NUM_THREADS = "ALL_CPUS")
)

get_dem(
  aoi,
  ...,
  start_date = NULL,
  end_date = NULL,
  pixel_x_size = 30,
  pixel_y_size = 30,
  asset_names = rsi::dem_band_mapping$planetary_computer_v1$`cop-dem-glo-30`,
  stac_source = attr(asset_names, "stac_source"),
  collection = attr(asset_names, "collection_name"),
  query_function = attr(asset_names, "query_function"),
  download_function = attr(asset_names, "download_function"),
  sign_function = attr(asset_names, "sign_function"),
  rescale_bands = FALSE,
  item_filter_function = NULL,
  mask_band = NULL,
  mask_function = NULL,
```

```
  output_filename = paste0(proceduralnames::make_english_names(1), ".tif"),
  composite_function = "max",
  limit = 999,
  gdalwarp_options = c("-r", "bilinear", "-multi", "-overwrite", "-co",
    "COMPRESS=DEFLATE", "-co", "PREDICTOR=2", "-co", "NUM_THREADS=ALL_CPUS"),
 gdal_config_options = c(VSI_CACHE = "TRUE", GDAL_CACHEMAX = "30%", VSI_CACHE_SIZE =
  "10000000", GDAL_HTTP_MULTIPLEX = "YES", GDAL_INGESTED_BYTES_AT_OPEN = "32000",
    GDAL_DISABLE_READDIR_ON_OPEN = "EMPTY_DIR", GDAL_HTTP_VERSION = "2",
    GDAL_HTTP_MERGE_CONSECUTIVE_RANGES = "YES", GDAL_NUM_THREADS = "ALL_CPUS")
)
```

Arguments

`aoi`	An sf(c) object outlining the area of interest to get imagery for. Will be used to get the bounding box used for calculating metrics and the output data's CRS.
`start_date, end_date`	Character of length 1: The first and last date, respectively, of imagery to include in metrics calculations. Should be in YYYY-MM-DD format.
`pixel_x_size, pixel_y_size`	Numeric of length 1: size of pixels in x-direction (longitude / easting) and y-direction (latitude / northing).
`asset_names`	The names of the assets to download. If this vector has names, then the names of the vector are assumed to be the names of assets on the STAC server, which will be renamed to the elements of the vector in the final output.
`stac_source`	Character of length 1: the STAC URL to download imagery from.
`collection`	Character of length 1: the STAC collection to download images from.
`...`	Passed to `item_filter_function`.
`query_function`	A function that takes the output from `rstac::stac_search()` and executes the request. See `rsi_query_api()` and the `query_function` slots of sentinel1_band_mapping, sentinel2_band_mapping, and landsat_band_mapping.
`download_function`	A function that takes the output from `query_function` and downloads the assets attached to those items. See `rsi_download_rasters()` for an example.
`sign_function`	A function that takes the output from `query_function` and signs the item URLs, if necessary.
`rescale_bands`	Logical of length 1: If the STAC collection implements the `raster` STAC extension, and that extension includes `scale` and `offset` values, should this function attempt to automatically rescale the downloaded data?
`item_filter_function`	A function that takes the outputs of `query_function` (usually a `STACItemCollection`) and `...` and returns a filtered `STACItemCollection`. This is used, for example, to only download images from specific Landsat platforms.
`mask_band`	Character of length 1: The name of the asset in your STAC source to use to mask the data. Set to `NULL` to not mask. See the `mask_band` slots of sentinel1_band_mapping, sentinel2_band_mapping, and landsat_band_mapping.

mask_function
: A function that takes a raster and returns a boolean raster, where TRUE pixels will be preserved and FALSE or NA pixels will be masked out. See sentinel2_mask_function().

output_filename
: The filename to write the output raster to. If composite_function is NULL, item datetimes will be appended to this in order to create unique filenames. If items do not have datetimes, a sequential ID will be appended instead.

composite_function
: Character of length 1: The name of a function used to combine downloaded images into a single composite (i.e., to aggregate pixel values from multiple images into a single value). Must be one of of "sum", "mean", "median", "min", "max". Set to NULL to not composite (i.e., to rescale and save each individual file independently).

limit
: an integer defining the maximum number of results to return. If not informed, it defaults to the service implementation.

gdalwarp_options
: Options passed to gdalwarp through the options argument of sf::gdal_utils(). The same set of options are used for all downloaded data and the final output images; this means that some common options (for instance, PREDICTOR=3) may cause errors if bands are of varying data types.

gdal_config_options
: Options passed to gdalwarp through the config_options argument of sf::gdal_utils().

platforms
: The names of Landsat satellites to download imagery from. These do not correspond to the platforms column in spectral_indices(); the default argument of c("landsat-9", "landsat-8") corresponds to the Landsat-OLI value in that column.

Value

output_filename, unchanged.

Usage Tips

It's often useful to buffer your aoi object slightly, on the order of 1-2 cell widths, in order to ensure that data is downloaded for your entire AOI even after accounting for any reprojection needed to compare your AOI to the data on the STAC server.

These functions allow for parallelizing downloads via future::plan(), and for user-controlled progress updates via progressr::handlers(). If there are fewer images to download than asset_names, then this function uses lapply() to iterate through images and future.apply::future_mapply() to iterate through downloading each asset. If there are more images than assets, this function uses future.apply::future_lapply() to iterate through images.

Downloading from Planetary Computer

Certain data sets in Planetary Computer require providing a subscription key. Even for non-protected data sets, providing a subscription key grants you higher rate limits and faster downloads. As such, it's a good idea to request a Planetary Computer account, then generate a subscription key. If you set the rsi_pc_key environment variable to your key (either primary or secondary; there is no difference), rsi will automatically use this key to sign all requests against Planetary Computer.

There are currently some challenges with certain Landsat images in Planetary Computer; please see https://github.com/microsoft/PlanetaryComputer/discussions/101 for more information on these images and their current status. These files may cause data downloads to fail.

Compositing

This function can either download all data that intersects with your spatiotemporal AOI as multiple files (if `composite_function = NULL`), or can be used to rescale band values, apply a mask function, and create a composite from the resulting files in a single function call. Each of these steps can be skipped by passing `NULL` to the corresponding argument.

Masks are applied to each downloaded asset separately. Rescaling is applied to the final composite after images are combined.

A number of the steps involved in creating composites – rescaling band values, running the mask function, masking images, and compositing images – currently rely on the `terra` package for raster calculations. This means creating larger composites, either in geographic or temporal dimension, may cause errors. It can be a good idea to tile your `aoi` using `sf::st_make_grid()` and iterate through the tiles to avoid these errors (and to make it easier to interrupt and restart a download job).

Rescaling

If `rescale_bands` is `TRUE`, then this function is able to use the `scale` and `offset` values in the `bands` field of the `raster` STAC extension. This was implemented originally to work with the Landsat collections in the Planetary Computer STAC catalogue, but hopefully will work automatically with other data sources as well. Note that Sentinel-2 data typically doesn't use this STAC extension, and so the returned data is typically not re-scaled; divide the downloaded band values by 10000 to get reflectance values in the expected values.

Sentinel-1 Data

The `get_sentinel1_imagery()` function is designed to download Sentinel-1 data from the Microsoft Planetary Computer STAC API. Both the GRD and RTC Sentinel-1 collections are supported. To download RTC data, set `collection` to `sentinel-1-rtc`, and supply your subscription key as an environment variable named `rsi_pc_key` (through, e.g., `Sys.setenv()` or your `.Renviron` file).

AlOS PALSAR Data

The `get_alos_palsar_imagery()` function is designed to download ALOS PALSAR annual mosaic data from the Microsoft Planetary Computer STAC API. Data are returned as a digital number (which is appropriate for some applications and indices). To convert to backscatter (decibels) use the following formula: `10 * log10(dn) - 83.0` where dn is the radar band in digital number.

Examples

```
aoi <- sf::st_point(c(-74.912131, 44.080410))
aoi <- sf::st_set_crs(sf::st_sfc(aoi), 4326)
aoi <- sf::st_buffer(sf::st_transform(aoi, 5070), 100)

get_stac_data(aoi,
```

```
  start_date = "2022-06-01",
  end_date = "2022-06-30",
  pixel_x_size = 30,
  pixel_y_size = 30,
  asset_names = c(
    "red", "blue", "green"
  ),
  stac_source = "https://planetarycomputer.microsoft.com/api/stac/v1/",
  collection = "landsat-c2-l2",
  query_function = rsi_query_api,
  sign_function = sign_planetary_computer,
  mask_band = "qa_pixel",
  mask_function = landsat_mask_function,
  item_filter_function = landsat_platform_filter,
  platforms = c("landsat-9", "landsat-8")
)

# or, mostly equivalently (will download more bands):
landsat_image <- get_landsat_imagery(
  aoi,
  start_date = "2022-06-01",
  end_date = "2022-08-30"
)
```

`landsat_band_mapping` *Landsat band mapping*

Description

This object is a named list of character vectors, with names corresponding to Landsat band names and values corresponding to band names in `spectral_indices`.

Usage

```
landsat_band_mapping
```

Format

An object of class `list` of length 1.

Details

Band mapping objects:

These objects are semi-standardized sets of metadata which provide all the necessary information for downloading data from a given STAC server. The object itself is list of character vectors, whose names represent asset names on a given STAC server and whose values represent the corresponding standardized band name from the Awesome Spectral Indices project. In addition to this data, these vectors usually have some of (but not necessarily all of) the following attributes:

- `stac_source`: The URL for the STAC server this metadata corresponds to.
- `collection_name`: The default STAC collection for this data source.
- `download_function`: The function to be used to download assets from the STAC server.
- `mask_band`: The name of the asset on this server to be used for masking images.
- `mask_function`: The function to be used to mask images downloaded from this server.

`landsat_mask_function` *Create a Landsat mask raster from the QA band*

Description

Create a Landsat mask raster from the QA band

Usage

```
landsat_mask_function(raster, include = c("land", "water", "both"))
```

Arguments

`raster`	The QA band of a Landsat image
`include`	Include pixels that represent land, water, or both? Passing `c("land", "water")` is identical to passing `"both"`.

Value

A boolean raster to be used to mask a Landsat image

Examples

```
aoi <- sf::st_point(c(-74.912131, 44.080410))
aoi <- sf::st_set_crs(sf::st_sfc(aoi), 4326)
aoi <- sf::st_buffer(sf::st_transform(aoi, 5070), 100)

landsat_image <- get_landsat_imagery(
  aoi,
  start_date = "2022-06-01",
  end_date = "2022-08-30",
  mask_function = landsat_mask_function
)
```

landsat_platform_filter *Filter Landsat features to only specific platforms*

Description

Filter Landsat features to only specific platforms

Usage

```
landsat_platform_filter(items, platforms)
```

Arguments

items	A STACItemCatalog containing some number of features
platforms	A vector of acceptable platforms, for instance landsat-9. Note that this refers to satellite names, and *not* to platforms in spectral_indices().

Value

A STACItemCollection.

Examples

```
aoi <- sf::st_point(c(-74.912131, 44.080410))
aoi <- sf::st_set_crs(sf::st_sfc(aoi), 4326)
aoi <- sf::st_buffer(sf::st_transform(aoi, 5070), 100)

landsat_image <- get_landsat_imagery(
  aoi,
  start_date = "2022-06-01",
  end_date = "2022-08-30",
  item_filter_function = landsat_platform_filter
)
```

rsi_download_rasters *Download specific assets from a set of STAC items*

Description

Download specific assets from a set of STAC items

Usage

```
rsi_download_rasters(
  items,
  aoi,
  asset_names,
  sign_function = NULL,
  merge = FALSE,
  gdalwarp_options = c("-r", "bilinear", "-multi", "-overwrite", "-co",
    "COMPRESS=DEFLATE", "-co", "PREDICTOR=2", "-co", "NUM_THREADS=ALL_CPUS"),
 gdal_config_options = c(VSI_CACHE = "TRUE", GDAL_CACHEMAX = "30%", VSI_CACHE_SIZE =
  "10000000", GDAL_HTTP_MULTIPLEX = "YES", GDAL_INGESTED_BYTES_AT_OPEN = "32000",
    GDAL_DISABLE_READDIR_ON_OPEN = "EMPTY_DIR", GDAL_HTTP_VERSION = "2",
    GDAL_HTTP_MERGE_CONSECUTIVE_RANGES = "YES", GDAL_NUM_THREADS = "ALL_CPUS",
  GDAL_HTTP_USERAGENT = "rsi (https://permian-global-research.github.io/rsi/)"),
  ...
)
```

Arguments

items	A StacItemCollection object, as returned by rsi_query_api().
aoi	Either an sf(c) object outlining the area of interest to get imagery for, or a bbox image containing the bounding box of your AOI.
asset_names	The names of the assets to download. If this vector has names, then the names of the vector are assumed to be the names of assets on the STAC server, which will be renamed to the elements of the vector in the final output.
sign_function	A function that takes the output from query_function and signs the item URLs, if necessary.
merge	Logical: for each asset, should data from multiple items be merged into a single downloaded file? If TRUE, this returns a single file for each asset, which has been merged via gdalwarp. No resampling or compositing is performed, but rather each pixel uses the last data downloaded. This is fast, but precludes per-item masking and compositing. If FALSE, each asset from each item is saved as a separate file.
gdalwarp_options	Options passed to gdalwarp through the options argument of sf::gdal_utils(). The same set of options are used for all downloaded data and the final output images; this means that some common options (for instance, PREDICTOR=3) may cause errors if bands are of varying data types.
gdal_config_options	Options passed to gdalwarp through the config_options argument of sf::gdal_utils().
...	Passed to item_filter_function.

Value

A data frame where columns correspond to distinct assets, rows correspond to distinct items, and cells contain file paths to the downloaded data.

Examples

```
aoi <- sf::st_point(c(-74.912131, 44.080410))
aoi <- sf::st_set_crs(sf::st_sfc(aoi), 4326)
aoi <- sf::st_buffer(sf::st_transform(aoi, 5070), 100)

landsat_image <- get_landsat_imagery(
  aoi,
  start_date = "2022-06-01",
  end_date = "2022-08-30",
  download_function = rsi_download_rasters
)
```

---

rsi_query_api *Query a STAC API using a specific spatiotemporal area of interest*

Description

This function is the default method used to retrieve lists of items to download for all the collections and endpoints supported by rsi. It will likely work for any other STAC APIs of interest.

Usage

```
rsi_query_api(bbox, stac_source, collection, start_date, end_date, limit, ...)
```

Arguments

bbox	An sfc object representing the spatial bounding box of your area of interest. This must be in EPSG:4326 coordinates (and, if this function is called from within `get_stac_data()`, it will be)
stac_source	Character of length 1: the STAC URL to download imagery from.
collection	Character of length 1: the STAC collection to download images from.
start_date, end_date	Character strings of length 1 representing the boundaries of your temporal range of interest, in RFC-3339 format. Set either argument to `..` to use an open interval; set `start_date` to `NULL` to not pass a temporal range of interest (which may cause errors with some APIs). If this function is called from within `get_stac_data()`, the inputs to `start_date` and `end_date` will have already been processed to try and force RFC-3339 compliance.
limit	an `integer` defining the maximum number of results to return. If not informed, it defaults to the service implementation.
...	Ignored by this function. Arguments passed to `get_stac_data()` via `...` will be available (unchanged) in this function

Details

You can pass your own query functions to `get_stac_data()` and its variants. This is the best way to perform more complex queries, for instance if you need to provide authentication to get the list of items (not just the assets) available for your AOI, or to perform cloud filtering prior to downloading assets.

Value

A StacItemCollection object.

Examples

```
aoi <- sf::st_point(c(-74.912131, 44.080410))
aoi <- sf::st_set_crs(sf::st_sfc(aoi), 4326)
aoi <- sf::st_buffer(sf::st_transform(aoi, 5070), 100)

landsat_image <- get_landsat_imagery(
  aoi,
  start_date = "2022-06-01",
  end_date = "2022-08-30",
  query_function = rsi_query_api
)
```

`sentinel1_band_mapping` *Sentinel-1 band mapping*

Description

This object is a named list of character vectors, with names corresponding to Sentinel-1 band names and values corresponding to band names in `spectral_indices`.

Usage

```
sentinel1_band_mapping
```

Format

An object of class `list` of length 1.

Details

Band mapping objects:

These objects are semi-standardized sets of metadata which provide all the necessary information for downloading data from a given STAC server. The object itself is list of character vectors, whose names represent asset names on a given STAC server and whose values represent the corresponding standardized band name from the Awesome Spectral Indices project. In addition to this data, these vectors usually have some of (but not necessarily all of) the following attributes:

- stac_source: The URL for the STAC server this metadata corresponds to.
- collection_name: The default STAC collection for this data source.
- download_function: The function to be used to download assets from the STAC server.
- mask_band: The name of the asset on this server to be used for masking images.
- mask_function: The function to be used to mask images downloaded from this server.

`sentinel2_band_mapping` *Sentinel-2 band mapping*

Description

This object is a named list of character vectors, with names corresponding to Sentinel-2 band names and values corresponding to band names in `spectral_indices`.

Usage

```
sentinel2_band_mapping
```

Format

An object of class `list` of length 3.

Details

Band mapping objects:

These objects are semi-standardized sets of metadata which provide all the necessary information for downloading data from a given STAC server. The object itself is list of character vectors, whose names represent asset names on a given STAC server and whose values represent the corresponding standardized band name from the Awesome Spectral Indices project. In addition to this data, these vectors usually have some of (but not necessarily all of) the following attributes:

- stac_source: The URL for the STAC server this metadata corresponds to.
- collection_name: The default STAC collection for this data source.
- download_function: The function to be used to download assets from the STAC server.
- mask_band: The name of the asset on this server to be used for masking images.
- mask_function: The function to be used to mask images downloaded from this server.

sentinel2_mask_function

Create a Sentinel-2 mask raster from the SCL band

Description

Create a Sentinel-2 mask raster from the SCL band

Usage

```
sentinel2_mask_function(raster)
```

Arguments

raster	The SCL band of a Sentinel-2 image

Value

A boolean raster to be used to mask a Sentinel-2 image

Examples

```
aoi <- sf::st_point(c(-74.912131, 44.080410))
aoi <- sf::st_set_crs(sf::st_sfc(aoi), 4326)
aoi <- sf::st_buffer(sf::st_transform(aoi, 5070), 100)

sentinel2_image <- get_sentinel2_imagery(
  aoi,
  start_date = "2022-06-01",
  end_date = "2022-08-30",
  mask_function = sentinel2_mask_function
)
```

sign_planetary_computer

Sign STAC items retrieved from the Planetary Computer

Description

Sign STAC items retrieved from the Planetary Computer

Usage

```
sign_planetary_computer(items, subscription_key = Sys.getenv("rsi_pc_key"))
```

Arguments

items	A STACItemCollection, as returned by rsi_query_api.
subscription_key	Optionally, a subscription key associated with your Planetary Computer account. At the time of writing, this is required for downloading Sentinel 1 RTC products, as well as NAIP imagery. This key will be automatically used if the environment variable rsi_pc_key is set.

Value

A STACItemCollection object with signed assets url.

Examples

```
aoi <- sf::st_point(c(-74.912131, 44.080410))
aoi <- sf::st_set_crs(sf::st_sfc(aoi), 4326)
aoi <- sf::st_buffer(sf::st_transform(aoi, 5070), 100)

landsat_image <- get_landsat_imagery(
  aoi,
  start_date = "2022-06-01",
  end_date = "2022-08-30",
  sign_function = sign_planetary_computer
)
```

---

spectral_indices *Get a data frame of spectral indices*

Description

This function returns a data frame of spectral indices, from the awesome-spectral-indices repository.

Usage

```
spectral_indices(
  ...,
  url = spectral_indices_url(),
  download_indices = NULL,
  update_cache = NULL
)
```

Arguments

`...`	These dots are for future extensions and must be empty.
`url`	The URL to download spectral indices from. If the option `rsi_url` is set, that value will be used; otherwise, if the environment variable `rsi_url` is set, that value will be used; otherwise, the list at https://github.com/awesome-spectral-indices/awesome-spectral-indices will be used.
`download_indices`	Logical: should this function download indices? If `NULL`, this function will only download indices if the cache will be updated. If `TRUE`, this function will attempt to download indices no matter what. If `FALSE`, either cached or package indices will be used.
`update_cache`	Logical: should cached indices be updated? If `NULL`, cached values will be updated if the cache is older than a day. If `TRUE`, the cache will be updated, if `FALSE` it will not.

Value

A tibble::tibble with nine columns, containing information about spectral indices.

Source

https://github.com/awesome-spectral-indices/awesome-spectral-indices

Examples

```
spectral_indices()
```

`spectral_indices_url` *Get the URL to download spectral indices from*

Description

Get the URL to download spectral indices from

Usage

```
spectral_indices_url()
```

Value

A URL to download indices from.

Examples

```
spectral_indices_url()
```

stack_rasters *Create and save a multi-band output raster by combining input rasters*

Description

This function creates an output raster that "stacks" all the bands of its input rasters, as though they were loaded one after another into a GIS. It does this by first constructing a GDAL virtual raster, or "VRT", and then optionally uses GDAL's warper to convert this VRT into a standalone file. The VRT is fast to create and does not require much space, but does require the input rasters not be moved or altered. Creating a standalone raster from this file may take a long time and a large amount of disk space.

Usage

```
stack_rasters(
  rasters,
  output_filename,
  ...,
  resolution,
  extent,
  reference_raster = 1,
  resampling_method = "bilinear",
  band_names,
  check_crs = TRUE,
  gdalwarp_options = c("-multi", "-overwrite", "-co", "COMPRESS=DEFLATE", "-co",
    "PREDICTOR=2", "-co", "NUM_THREADS=ALL_CPUS"),
 gdal_config_options = c(VSI_CACHE = "TRUE", GDAL_CACHEMAX = "30%", VSI_CACHE_SIZE =
  "10000000", GDAL_HTTP_MULTIPLEX = "YES", GDAL_INGESTED_BYTES_AT_OPEN = "32000",
    GDAL_DISABLE_READDIR_ON_OPEN = "EMPTY_DIR", GDAL_HTTP_VERSION = "2",
    GDAL_HTTP_MERGE_CONSECUTIVE_RANGES = "YES", GDAL_NUM_THREADS = "ALL_CPUS")
)
```

Arguments

rasters	A list of rasters to combine into a single multi-band raster, as character file paths to files that can be read by terra::rast(). Rasters will be "stacked" upon one another, preserving values. They must share CRS.
output_filename	The location to save the final "stacked" raster. If this filename has a "vrt" extension as determined by tools::file_ext(), then this function exits after creating a VRT; otherwise, this function will create a VRT and then use sf::gdal_utils("warp") to convert the VRT into another format.
...	These dots are for future extensions and must be empty.
resolution	Numeric of length 2, representing the target X and Y resolution of the output raster. If only a single value is provided, it will be used for both X and Y resolution; if more than 2 values are provided, an error is thrown.

`extent` Numeric of length 4, representing the target xmin, ymin, xmax, and ymax values of the output raster (its bounding box), in that order.

`reference_raster`
The position (index) of the raster in `rasters` to take extent, resolution, and CRS information from. No reprojection is done. If `resolution` or `extent` are provided, they override the values from the reference raster.

`resampling_method`
The method to use when resampling to different resolutions in the VRT.

`band_names` Either a character vector of band names, or a function that when given a character vector of band names, returns a character vector of the same length containing new band names.

`check_crs` Logical: Should this function check that all `rasters` share the same CRS? Set to `FALSE` only if you are entirely confident that rasters have equivalent CRS definitions, but not identical WKT strings.

`gdalwarp_options`
Options passed to `gdalwarp` through the `options` argument of `sf::gdal_utils()`. This argument is ignored (with a warning) if `output_filename` is a VRT.

`gdal_config_options`
Options passed to `gdalwarp` through the `config_options` argument of `sf::gdal_utils()`. This argument is ignored (with a warning) if `output_filename` is a VRT.

Value

`output_filename`, unchanged.

Examples

```
stack_rasters(
  list(
    system.file("rasters/dpdd.tif", package = "rsi"),
    system.file("rasters/example_sentinel1.tif", package = "rsi")
  ),
  tempfile(fileext = ".vrt")
)
```

Index