Package: caladaptr (via r-universe)

March 5, 2025

Type Package

Title Import Climate Data from Cal-Adapt via the API

Version 0.6.8

Date 2022-11-29

BugReports https://github.com/ucanr-igis/caladaptr/issues

URL https://ucanr-igis.github.io/caladaptr

Description Base functions for importing climate data using the
Cal-Adapt API <https://cal-adapt.org/>.

License GPL (>= 3)

Encoding UTF-8

LazyData true

Imports backports, crayon, curl, DBI, dbplyr, digest, dplyr,
fastmatch, geojsonsf, httr, lifecycle, magrittr, purrr,
RSQLite, sf, shiny, stars, tibble, tmap, units, utils, zip

RoxygenNote 7.2.1

Suggests chillR, cubelyr, ggplot2, lubridate, DiagrammeR, httptest,
testthat, knitr, parallel, rmarkdown, roxygen2

Depends R (>= 3.6)

VignetteBuilder knitr

Config/testthat/edition 3

Config/pak/sysreqs libgdal-dev gdal-bin libgeos-dev make libicu-dev
libpng-dev libxml2-dev libssl-dev libproj-dev libsqlite3-dev
libudunits2-dev zlib1g-dev

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

RemoteUrl https://github.com/ucanr-igis/caladaptr

RemoteRef HEAD

RemoteSha 0b8608fa7f9a3bebe7a042ace9701256dd5f651a

Contents

aoipreset_types 3
bbox_resize 3
ca_aoipreset_geom 4
ca_apireq 5
ca_baseurl 5
ca_biggeom_blocks 6
ca_catalog_fetch 7
ca_catalog_rs 7
ca_catalog_search 8
ca_cvar 9
ca_dates 10
ca_db_indices 10
ca_db_info 11
ca_db_read 12
ca_example_apireq 13
ca_gcm 14
ca_getcache 14
ca_getrst_stars 15
ca_getvals_db 16
ca_getvals_tbl 19
ca_livneh 20
ca_locagrid_geom 20
ca_loc_aoipreset 21
ca_loc_pt 22
ca_loc_sf 22
ca_options 23
ca_period 24
ca_preflight 24
ca_scenario 26
ca_settings 26
ca_slug 27
ca_stars_6d 28
ca_stars_index 28
ca_stars_mosaic 29
ca_stars_read 30
ca_years 31
cvars 32
format.ca_apireq 32
format.ca_db_info 33
gcms 33
periods 34
plot.ca_apireq 35
print.ca_apireq 35
print.ca_db_info 36
scenarios 36

Index **37**

`aoipreset_types` *Area-of-interest presets*

Description

Area of interest presets

The field(s) each AOI Preset provides to identify features

Values that can be used to identify features in AOI Preset

Usage

```
data(aoipreset_types)

data(aoipreset_idflds)

data(aoipreset_idval)
```

Format

An character vector with the names of area-of-interest presets

Named list with with one element per preset type

Named list of data frames, with with one element per preset type

Functions

- `aoipreset_idflds`: AOI Preset id fields
- `aoipreset_idval`: Values that can be used to identify features

`bbox_resize` *Resize a bounding box object*

Description

Resize a bounding box object by a scale factor or buffer distance

Usage

```
bbox_resize(x, scale = NULL, buff = NULL)
```

Arguments

`x`	bbox object
`scale`	scale factor
`buff`	buffer distance

Details

You can resize a bounding box by passing a value for `scale` or `buff`, but not both.

Use `scale` to resize the bounding box with a scale factor. Values of `scale` < 1 will result in a smaller bounding box, values > 1 will result in a larger bounding box. The centroid will remain the same. If you pass two values for `scale`, they'll be used to scale the x and y dimensions respectively

Use `buff` to resize the bounding box with a fixed distance. `buff` should be in a map units. Values of `buff` < 0 will result in a smaller bounding box. Values > 0 will result in a larger bounding box. If you pass two values for `buff`, they'll be used to buffer the x and y dimensions respectively

`ca_aoipreset_geom` *Get the geometry of an AOI Preset area*

Description

Get the geometry of an AOI Preset area

Usage

```
ca_aoipreset_geom(aoipreset, quiet = FALSE)
```

Arguments

`aoipreset`	The name of a AOI preset
`quiet`	Suppress messages

Details

This retrieves the geometry (i.e., boundaries) for one of Cal-Adapt's AOI Presets. If the spatial layer has not already been downloaded, it will be downloaded (from `https://github.com/ucanr-igis/caladaptr/tree/master/aoipreset_geoms`) and saved in the local cache directory as a GeoPackage. The default local cache directory is buried in the current user's 'AppData' folder. To put the GeoPackages in an easier-to-find location, use `ca_setcache`.

Value

A simple feature data frame

See Also

`aoipreset_types`, `ca_getcache`

`ca_apireq` *Creates a new API request object*

Description

Creates a new API request object

Usage

```
ca_apireq(
  loc = NA,
  dates = NA,
  gcm = NA,
  scenario = NA,
  period = NA,
  cvar = NA,
  livneh = NA,
  slug = NA,
  options = NA
)
```

Arguments

`loc`	A location object (see Details)
`dates`	A dates object (see Details)
`gcm`	A vector of GCM abbreviations (see ‘gcms‘)
`scenario`	A vector of scenario names (see ‘scenarios‘)
`period`	A vector of period names (see ‘periods‘)
`cvar`	A vector of climate variables (see ‘climvars‘)
`livneh`	Use Livneh dataset, logical
`slug`	A vector of slugs
`options`	A list of options for querying the API

`ca_baseurl` *Cal-Adapt base URL*

Description

Base URL for all calls to the Cal-Adapt API

Usage

```
data(ca_baseurl)
```

Format

An character vector

`ca_biggeom_blocks` *Split a large geom into blocks small enough to ask the API for rasters*

Description

Split a large geom into blocks small enough to ask the API for rasters

Usage

```
ca_biggeom_blocks(x)
```

Arguments

`x`	A big geom

Details

The Cal-Adapt API has a limit of around 20,000 mi^2 as the maximum area for which you can download a raster. This function will take a sf data frame larger than this and return blocks that cover the same extent. Subsequently you can download rasters for the individual blocks and mosaic them into the full area using ca_stars_mosaic.

Note while this function can help you work around the maximum area you can download tifs via the API, it won't help you get spatially aggregated values from a large area using the API. For that, you would need to a) use this function to download rasters, b) mosaic them, and c) do a spatial aggregration on the large area-of-interest.

If your study areas encompasses the entire Cal-Adapt coverage area, you'd be better off downloading the individual rasters from the [Cal-Adapt Data Server](http://albers.cnr.berkeley.edu/data/).

Value

A polygon simple feature data frame covering the same extent as `x`

See Also

`ca_getrst_stars`, `ca_stars_read`

`ca_catalog_fetch` *Fetch a new copy of the Cal-Adapt raster series catalog*

Description

Fetch a new copy of the Cal-Adapt raster series catalog

Usage

```
ca_catalog_fetch(quiet = FALSE, save_to_cache = TRUE)
```

Arguments

`quiet`	Suppress messages, logical
`save_to_cache`	Save the catalog to the cache directory, logical

Details

This function will download a list of all raster series available through the Cal-Adapt API. If `save_to_cache = TRUE`, the catalog will be saved as a csv file in the caladaptR's cache folder and used for subsequent calls to `ca_catalog_rs`.

A copy of the raster series catalog is also included with caladaptR. You can run `ca_catalog_fetch()` to update the catalog when new raster series are published on Cal-Adapt. The best way to find out when new data are published on Cal-Adapt is to subscribe to the Cal-Adapt newsletter.

Value

The raster series catalog as a tibble

See Also

`ca_catalog_rs`, `ca_getcache`, `ca_setcache`

`ca_catalog_rs` *Get the Cal-Adapt raster series data catalog*

Description

Get a local copy of the Cal-Adapt raster series data catalog

Usage

```
ca_catalog_rs(quiet = FALSE)
```

Arguments

`quiet`	Suppress messages

Details

This retrieves a local copy of the Cal-Adapt 'catalog' of raster series available through the Cal-Adapt API. A copy of the catalog comes with caladaptR. You can also fetch a new copy using ca_catalog_fetch.

Value

A tibble with columns of information about the raster series available through the Cal-Adapt API.

See Also

`ca_catalog_fetch`, `ca_getcache`

`ca_catalog_search` *Search the Cal-Adapt raster series data catalog*

Description

Search the Cal-Adapt raster series data catalog

Usage

```
ca_catalog_search(x, keep_together = FALSE, quiet = FALSE)
```

Arguments

`x`	text to search for
`keep_together`	treat x as a phrase, logical
`quiet`	suppress messages

Details

This function can be used to search the local copy of the Cal-Adapt raster series data catalog, and view the properties of the matching results. Searched fields include the dataset name and slug. If `keep_together = TRUE`, the search text will be treated as a phrase, otherwise the words in x will be searched for separately. Records have to match all terms to be returned.

For an online search tool, click the 'Filters' button on `https://api.cal-adapt.org/api/series/`.

Value

A tibble with information about the found raster series

See Also

ca_catalog_rs, ca_catalog_fetch

Examples

```
## Not run:
## Search for a slug
ca_catalog_search("pr_day_gridmet")

## Search for keywords
ca_catalog_search("evapotranspiration year")

## Search for phrase
ca_catalog_search("Livneh VIC", keep_together = TRUE)

## End(Not run)
```

---

ca_cvar — *Add climate variable(s) to a Cal-Adapt API request*

Description

Specifies climate variable(s) a Cal-Adapt API call should retrieve

Usage

```
ca_cvar(x = ca_apireq(), cvar)
```

Arguments

x	Cal-Adapt API request
cvar	Climate variable

Details

For valid options for `cvar`, see cvars.

Notes:

1) 'climate variables' refers to both the variables returned by global circulation models (e.g., temperature, precipitation), as well as variables derived by additional models (e.g., evapo-transpiration)

2) Not all climate variables are available for all climate models, temporal periods, and date ranges.

`ca_dates` *Adds a start and end date of a Cal-Adapt API call*

Description

Specifies the start and end date of a Cal-Adapt API call

Usage

```
ca_dates(x = ca_apireq(), start, end)
```

Arguments

`x`	Cal-Adapt API request
`start`	start date entered as a character *yyyy-mm-dd* or Date object
`end`	end date entered as a character *yyyy-mm-dd* or Date object

`ca_db_indices` *Add or delete indices*

Description

Add or delete indices

Usage

```
ca_db_indices(x, tbl, idx_fld_add = NULL, idx_fld_del = NULL, quiet = FALSE)
```

Arguments

`x`	Either a remote tibble or a SQLite database file name
`tbl`	The name of a table in the SQLite database
`idx_fld_add`	Fields in tbl to create an index for
`idx_fld_del`	Fields in tbl that have indices you'd like to delete
`quiet`	Suppress messages

Details

Database indices improve performance when you filter or sort rows based on field (column), and/or join tables based on a common field. By default, indices are not created when you download Cal-Adapt data into a SQLite database with `ca_getvals_db` (because they increase the size of the SQLite file). You can tell `ca_getvals_db` to create indices with the `indices` argument, or use `ca_db_indices` to create indices after data are downloaded.

`x` can be either a remote tibble returned by `ca_getvals_db`, or a SQLite database file name. `tbl` should be the name of a table in the database (i.e., the `db_tbl` argument you passed to `ca_getvals_db`. If you're not sure what the table names are in a database, run `ca_db_info`. Normally you would only add indices to a table that contains values from Cal-Adapt (there is no need to add indices to lookup tables). You can only add indices for one table at a time (but `idx_fld_add` can contain multiple field names).

Note that `ca_db_indices` can only create indices on a single field. To create composite indices you can run SQL expressions with the DBI package. Indices added by `ca_db_indices` will be named automatically.

For more details, see the vignette on querying large volumes of data: `vignette("large-queries", package = "caladaptr")`

Value

`x`

See Also

`ca_getvals_db`, `ca_db_info`

`ca_db_info` *View properties of a Cal-Adapt SQLlite database*

Description

View properties of a Cal-Adapt SQLlite database

Usage

```
ca_db_info(x)
```

Arguments

`x`	Either a Cal-Adapt values remote tibble or a SQLite database file name

Details

`x` can be either a remote tibble returned by `ca_getvals_db`, or a SQLite database file name.

Value

A list object with info about x, including the tables, fields, indices, and sql statements.

See Also

ca_getvals_db

---

ca_db_read *Load a Cal-Adapt SQLite database into R*

Description

Load a Cal-Adapt SQLite database into R

Usage

```
ca_db_read(
  x,
  val_tbl = NULL,
  join_lookup_tbls = TRUE,
  all_tables = FALSE,
  exclude_hash_tables = TRUE
)
```

Arguments

x	A Cal-Adapt values remote tibble or SQLite file name
val_tbl	The name of table which contains climate data, ignored if all_tables = TRUE
join_lookup_tbls	Join lookup tables if present in the remote tibble
all_tables	Return all tables, logical
exclude_hash_tables	Exclude tables that contain search hashes, ignored if all_tables = FALSE

Details

This will 'mount' a SQLite database created by ca_getvals_db, and return a remote tibble (i.e., a tibble connected to a database). x can be either a SQLite file name or a remote tibble returned by ca_getvals_db.

val_tbl should be the name of the table in the database that contains the climate values (i.e., the same table you specified when you ran ca_getvals_db). If val_tbl = NULL and a sidecar text file for the SQLite database exists, it will search the sidecar file for the name of a values table and use the first one. if join_lookup_tbls = TRUE and lookup tables were used when fetching the data, the remote tibble returned will be based on a SQL expression that joins the lookup tables to the values table. Joining lookup tables is only possible if a sidecar text file exists.

If all_tables = TRUE, a list of remote tibbles for all the tables in the database will be returned. In this case, val_tbl and join_lookup_tbls are ignored. If exclude_hash_tables = TRUE, tables that store search hashes will be excluded in the returned list.

Value

A remote tibble with climate values if all_tables = FALSE, otherwise a list of remote tibbles if all_tables = TRUE

See Also

`ca_getvals_db`, `ca_db_info`

`ca_example_apireq` *Sample API requests*

Description

Sample API requests

Usage

```
ca_example_apireq(x)
```

Arguments

`x`	The number of a sample API request to return

Details

These sample API requests can be used in demos, documentation, and tests. `x` should be an integer:

`x = 1`: Basic API request for Scripps data – one point, 4 CGMs, 20 years of annual data.

`x = 2`: Three Congressional districts, monthly data, 4 years

`x = 3`: sf data frame with one feature, 1 GCM, 1 scenario, 2 years of daily data

`x = 4`: sf data frame with two multipolygons, 1 GCM, 1 scenario, 20 years of annual data

`x = 5`: Livheh data, ten census tracts, 20 years of daily temp data, spatial aggregation mean

`x = 6`: Livheh data, five census tracts (including one from #5), 5 years of daily temp data, spatial aggregation = mean

`x = 7`: Basic API request for Scripps data – one point, 4 CGMs, 70 years of annual data.

`ca_gcm` *Adds GCM(s) to a Cal-Adapt API request*

Description

Specifies GCM(s) a Cal-Adapt API call should retrieve

Usage

```
ca_gcm(x = ca_apireq(), gcm)
```

Arguments

`x`	Cal-Adapt API request
`gcm`	Global Climate Model abbreviation

Details

For valid options for `gcm`, see `gcms`.

`ca_getcache` *Manage cache directory*

Description

View and set the directory for the data catalog

Usage

```
ca_getcache(quiet = TRUE)

ca_setcache(
  cache_dir = NULL,
  make_dir = TRUE,
  save = TRUE,
  reset = FALSE,
  quiet = FALSE
)
```

Arguments

`quiet`	Suppress messages
`cache_dir`	The directory for cached data
`make_dir`	Make the directory if needed, logical
`save`	Save the cache directory in the `.Renviron` file for persistence across R sessions
`reset`	Change to the default location

Details

caladaptr has the ability to store copies of objects downloaded from Cal-Adapt. An example of this would be the raster series data catalog and the geometries of AOI Presets.

NOTE: In general caladaptr does *not* cache climate data fetched from Cal-Adapt. Every time you call a function that fetches data (e.g., ca_getvals_tbl), data is retrieved fresh. The exception to this is ca_getvals_db, which has arguments you can pass to cache retrieved values into a local SQLite database to explicitly avoid downloading data twice.

The default location for the cache directory is given by `R_user_dir("caladaptr", "cache")`. A custom location can be set with `ca_setcache`.

Functions

- `ca_setcache()`: Set cache directory

See Also

ca_catalog_rs, ca_aoipreset_geom, ca_locagrid_geom

---

ca_getrst_stars — *Get cropped rasters*

Description

Download a cropped raster for an API request

Usage

```
ca_getrst_stars(
  x,
  out_dir = NULL,
  mask = TRUE,
  merge_geoms = FALSE,
  sidecar = TRUE,
  stop_on_err = TRUE,
  overwrite = FALSE,
  normalize_path = FALSE,
  debug = FALSE,
  quiet = FALSE,
  write_sidecar = deprecated()
)
```

Arguments

x	A Cal-Adapt API request
out_dir	Where the output TIF files should be written
mask	Mask pixels outside the location of interest with NA values

`merge_geoms`	Whether to merge geometries, see Details
`sidecar`	Save a small sidecar file with the TIF file containing additional attribute info
`stop_on_err`	Stop if the server returns an error
`overwrite`	Re-download and overwrite existing files
`normalize_path`	Expand and normalize output file names
`debug`	Print additional output at the console
`quiet`	Suppress messages
`write_sidecar`	Deprecated

Details

This will download time series cropped raster(s) for your study area, convert them to stars objects, and export them as tif files. If `mask = TRUE`, pixels values outside the area of interest will be set to `NA` (`mask` is ignored for point locations). To get a single raster per dataset that encompasses all the locations, pass `merge_geoms = TRUE`.

Note this will only work for areas-of-interest small enough for the Cal-Adapt API to handle (i.e., smaller than San Bernadino County). If you want to download rasters for a large area (e.g., the whole state of California) you're better off downloading NetCDF files from the Cal-Adapt data server.

If `sidecar = TRUE`, a small file with the same base name as the tif will be saved. This sidecar file contains attributes of a space-time-array not preserved by tifs. You can import the tif file back into R as a stars object with `ca_read_stars`.

This function merely downloads the cropped rasters to disk and returns the filenames. To work with cropped rasters as stars objects within R, import them using `ca_read_stars`. You can also import the TIF files with other packages or software.

Value

A vector of TIF file names. If `normalize_path = TRUE` the output file names will be expanded (absolute) and use standard slashes for the OS (see normalizePath).

See Also

`ca_read_stars`

`ca_getvals_db` *Write values from an API request to a local database*

Description

Write values from an API request to a local database

Usage

```
ca_getvals_db(
  x,
  db_fn,
  db_tbl,
  omit_col = NULL,
  indices = NULL,
  new_recs_only = TRUE,
  trans_len = 100,
  lookup_tbls = TRUE,
  lookup_ret_joined = TRUE,
  pause_n = 1000,
  pause_secs = 60,
  write_sidecar = TRUE,
  stop_on_err = TRUE,
  quiet = FALSE,
  debug = FALSE
)
```

Arguments

`x`	A Cal-Adapt API request
`db_fn`	File name of a SQLite database. See Details.
`db_tbl`	The name of a database table. See Details.
`omit_col`	Columns to exclude from the tibble
`indices`	Name of fields to index. See Details.
`new_recs_only`	Write new records only to the database. See Details.
`trans_len`	Number of APIs calls per write transaction. See Details.
`lookup_tbls`	Use lookup tables
`lookup_ret_joined`	Return a table with lookup table fields, ignored if lookup_tbls = FALSE. See Details.
`pause_n`	Number of API calls after which a built-in pause is triggered. See Details.
`pause_secs`	Number of seconds to pause. See Details.
`write_sidecar`	Save table metadata in a separate file. See Details.
`stop_on_err`	Stop if the server returns an error
`quiet`	Suppress messages
`debug`	Print additional output at the console

Details

`ca_getvals_db` fetches data from the Cal-Adapt API and writes the data to a SQLite database as they're received. This allows you to fetch relatively large volumes of data in the background, and potentially over multiple sessions as it will pick up where it left off if interrupted. Saving the values in a database also reduces the risk of exhausting your RAM.

Use `ca_getvals_db` to fetch large volumes of data (i.e., hundreds of thousands of values), or whenever you'd like to keep a local copy of the data. Note however for small amounts of data there is no advantage to putting it in a database as it will be slighly slower to retrieve and work with.

`db_fn` should be a file name with path to a SQLite database. A SQLIte database is a single file typically with a *.db* or *.sqlite* extension. If the database doesn't exist, it will be created. If it already exists, the new data will be added to it.

`db_tbl` should the name of a table within the database where the new data will be saved. The table name should not contain special characters and spaces are discouraged. If `new_recs_only = TRUE`, only new records will be added to the database.

`trans_len` defines the number of API calls per SQLite transaction (i.e. how many API calls of data to accumulate before doing a write operation to the database). This can speed things up. Set it to 0 to disable transactions.

If `lookup_tbls = TRUE`, the database will create lookup tables for categorical columns such as GCM, scenario, cvar, period, slug, etc. This can dramatically reduce the size of the SQLite database file and is generally recommended. id `lookup_ret_joined = TRUE`, the tibble returned will have the lookup tables joined (i.e., column names will be unaltered); if not the returned tibble will have id values for certain values. A small text file is created for each SQLite database containing the names of the tables and SQL statement to join them (read automatically by `ca_db_read`).

`indices` is a vector of column names in `db_tbl` that you'd like indexed (ignored if `lookup_tbls = FALSE`). Creating indices can improve the performance of filters and joins when you generate summaries, but at the cost of a larger database file and slightly slower write operations. Fields you can create indices on include `"feat_id"` (the location id value), `"cvar"`, `"gcm"`, `"scenario"`, `"period"`, `"slug"`, and `"spag"`.

Indices can also be added to a SQLite database after downloading is complete with ca_db_indices. For large queries (e.g. thousands of API calls), it is recommended to not build indices during the download process, and only add indices for those fields you plan to filter on or join during your analysis. You can view which indices exist with `ca_db_info`.

`pause_n` is the number of API calls after which a built-in pause of length `pause_secs` is triggered. This is intended to avoid disruption on the Cal-Adapt server. The maximum value for `pause_n` is 2500, and the minimum value for `pause_secs` is 30 seconds.

The returned tibble is linked to the SQLite datbase. For the most part you can use the same dplyr functions to manipulate the results, but to retrieve the actual values you need to use 'collect()'. For more info working with a linked database, see `https://dbplyr.tidyverse.org/articles/dbplyr.html`.

Value

A remote tibble linked to the SQLite database.

See Also

`ca_db_info`, `ca_db_indices`, `ca_db_read`,

`ca_getvals_tbl` *Get values from an API request object as a tibble*

Description

Get values from an API request object as a tibble

Usage

```
ca_getvals_tbl(
  x,
  quiet = FALSE,
  debug = FALSE,
  stop_on_err = TRUE,
  shiny_progress = NULL,
  omit_col = NULL,
  timeout = NULL
)
```

Arguments

`x`	A Cal-Adapt API request
`quiet`	Suppress messages
`debug`	Print additional output at the console
`stop_on_err`	Stop if the server returns an error
`shiny_progress`	A Shiny progress bar object, see Details.
`omit_col`	Columns to exclude from the tibble
`timeout`	Timeout limit in seconds

Details

`ca_getvals_tbl` fetches data via the Cal-Adapt API, returning a tibble. Everything is done in memory. To download Cal-Adapt into a local SQLite database, see `ca_getvals_db`. To download Cal-Adapt data as raster files, see `ca_getrst_stars`.

A default set of columns will be returned based on how the dataset is specified (i.e., by slug, cvar+scen+gcm+per, livneh, etc). Some columns can be omitted by passing column names to `col_omit`. Three columns that can never be omitted are `feat_id` (location id value), `dt` (date), and `val` (the actual climate values).

`timeout` set the longest amount of time before curl reports an error. The default is 10 seconds. Increase this if you experience timeout errors (which have been know to occur on ShinyApps.io perhaps due to server congestion).

Value

A tibble

See Also

ca_getvals_db, ca_getrst_stars

---

ca_livneh *Use Livneh data in a Cal-Adapt API call*

Description

Specify Livneh data should be retrieved in a Cal-Adapt API call

Usage

```
ca_livneh(x = ca_apireq(), livneh = TRUE)
```

Arguments

`x`	Cal-Adapt API request
`livneh`	Use Livneh data, logical

Details

Convenience function to create an API request object for one of the Livneh datasets

See Also

ca_catalog_rs

---

ca_locagrid_geom *Get the LOCA grid cells as a sf object*

Description

Get the geometry of the LOCA grid cells as a sf polygon object

Usage

```
ca_locagrid_geom(quiet = FALSE)
```

Arguments

`quiet`	Suppress messages

Details

This retrieves the geometry of the LOCA grid as a vector (polygon) layer. The cells in this grid represent the pixels for all the LOCA downscaled raster series on Cal-Adapt. A copy of the layer will be saved in the cache folder so it won't have to be downloaded more than once.

Value

A simple feature data frame

See Also

`ca_getcache`

`ca_loc_aoipreset` *Adds a preset location to a Cal-Adapt API request*

Description

Adds a preset location to a Cal-Adapt API request

Usage

```
ca_loc_aoipreset(x = ca_apireq(), type, idfld = "id", idval = NULL)
```

Arguments

`x`	A Cal-Adapt API request
`type`	The type of AOI preset (see Details)
`idfld`	The name of the field that identifies the desired locations
`idval`	The value(s) of idfld

Details

`type` specifies one of the preset areas of interest supported by the Cal-Adapt API. For valid values, view the built-in constant `aoipreset_types`.

`idfld` is the field which contains the values you want to use to select the preset areas. For a list of fields you can use for each AOI preset, see the built-in list `ca_aoipreset_idflds`.

`idval` are the value(s) you can use to select specific areas of interest. For a list of values, see the built-in list `ca_aoipreset_idval`. If `idval = NULL`, all areas will be used.

Note all of the AOI Presets supported by the Cal-Adapt API are *polygons*. This means in order to query values for these areas (i.e., with `ca_getvals_tbl`), you must also specify a spatial aggregation function using `ca_options`.

See Also

`ca_apireq`

ca_loc_pt — *Add point location(s) to a Cal-Adapt API request*

Description

Specifies point location(s) a Cal-Adapt API call should retrieve

Usage

```
ca_loc_pt(x = ca_apireq(), coords, id = NULL)
```

Arguments

`x`	A Cal-Adapt API request
`coords`	A two-column matrix or data frame
`id`	Unique id values for each point

Details

`coords` should be a two-column matrix or data frame with the first column containing the x (longitude) values of the points of interest, and the second column containing the y (latitude) values. Projected coordinates can not be used with this function (but see `ca_loc_sf`).

`id` should be vector that uniquely identify the points. If omitted, row numbers will be used.

See Also

`ca_loc_sf`, `ca_apireq`

ca_loc_sf — *Use a sf data frame as the location for a Cal-Adapt API request*

Description

Specifies a sf data frame as the location for a Cal-Adapt API request

Usage

```
ca_loc_sf(x = ca_apireq(), loc, idfld = NULL, idval = NULL, dTolerance = 0)
```

Arguments

`x`	A Cal-Adapt API request
`loc`	A simple feature data frame
`idfld`	The name of a column in loc containing unique values, or the name
`idval`	A vector of unique values
`dTolerance`	A numeric value used to simplify polgyons, see Details.

Details

loc should be a simple feature data frame with point or polygon features. The sf object should have a valid CRS, but does not have to be geographic. Both 'single' and 'multipart' polygons can be used, but only simple point features are supported. To convert a multipoint feature layer into a single point layer, use st_cast.

idfld should be the name of a column in loc containing unique values. When you fetch values from Cal-Adapt, this column will be returned in the results to help you join the values to other tables. Alternately, you can use idfld to pass the name of a new column, together with a vector of unique values in idval (one for each row in loc).

Note you can not use idval as a filter. If you want to filter the features of loc to query, use a filter expression as part of the value of loc (e.g., with filter or slice).

If loc is a polygon layer, you'll also need to specify how to spatially aggregate the queried values if a feature overlaps more than one pixel. See ca_options.

dTolerance is a value in decimal degrees used to simplify polygons. If dTolerance > 0, geos_unary{st_simplify} will be called to remove polygon nodes within dTolerance of another node, before fetching data. This can reduce the amount of spatial data that needs to be sent to the server, which can improve performance particularly when you have very fine grained polygons. Simplifying polygons can modify (generally reduce) the pixels that a polygon overlaps, so use with caution. dTolerance = 0.001 represents <100m on the ground within Cal-Adapt range of latitude.

---

ca_options *Add processing options to a Cal-Adapt API call*

Description

Specify processing options for a Cal-Adapt API call

Usage

```
ca_options(
  x = ca_apireq(),
  spatial_ag = c("none", "mean", "max", "median", "min", "sum")[1],
  temporal_ag = NA
)
```

Arguments

x	Cal-Adapt API request
spatial_ag	Spatial aggregation function for polygon locs.
temporal_ag	List object specifying unit of time and summary function(s) for temporal aggregation. NOT YET SUPPORTED

Details

spatial_ag is the name(s) of summary statistic(s) that will be used when querying polygon locations to aggregate the values of pixel that fall within the area of interest. Values can be mean, max, median, min, and sum. To get retrieve the individual values for all pixels without a summary, use the ca_getrst function.

When querying point locations (e.g., ca_loc_pt or ca_loc_zip, spatial_ag should be set to 'none'.

temporal_ag allows you to apply an *additional* temporal aggregation function, on top of any temporal aggregation the data are already summarized by (e.g., month or year). When querying climate layers that are already temporally aggregated, the unit of temporal aggregation must be a larger unit of time (e.g., you can't pull down annual average layers and then try to aggregate them by month).

ca_period *Assign temporal aggregation period to a Cal-Adapt API call*

Description

Add a temporal aggregation period to a Cal-Adapt API request

Usage

```
ca_period(x = ca_apireq(), period)
```

Arguments

x	Cal-Adapt API request
period	Period of temporal aggregation

Details

For valid options for period, run periods.

Notes:

ca_preflight *Run checks on an API request object*

Description

Run checks on an API request object

Usage

```
ca_preflight(
  x,
  slug_check = TRUE,
  date_check = TRUE,
  loc_check = TRUE,
  units_check = TRUE,
  spag_check = TRUE,
  check_for = c("getvals", "getrst"),
  quiet = FALSE,
  ignore_spag = deprecated()
)
```

Arguments

`x`	A Cal-Adapt API request
`slug_check`	Cross check the slug against the raster series catalog
`date_check`	Cross check the start and end date against the raster series catalog
`loc_check`	Check to make sure the location is within the Cal-Adapt coverage area
`units_check`	Check for consistent units
`spag_check`	Check spatial aggregation option
`check_for`	What to check for - getting values or getting rasters
`quiet`	Suppress messages
`ignore_spag`	Deprecated

Details

This function checks an Cal-Adapt API request for potential problems. It checks to make sure the request:

- is complete
- doesn't have conflicting elements
- specifies an existing datasets
- specifies a location within the Cal-Adapt coverage area
- specifies area-of-interest presets correctly
- has features that are not too large for the Cal-Adapt API
- doesn't mix datasets that have different units
- uses dates that fall within the Cal-Adapt time series
- includes a spatial aggregation function if needed

Most of the checks can be selectively disabled using arguments. `check_for` allows you to tailor checks for querying values and/or downloading rasters.

Value

TRUE if no messages are reported, else FALSE

See Also

ca_catalog_rs

---

ca_scenario — *Add emission scenario(s) to a Cal-Adapt API request*

Description

Specifies emission scenario(s) for retrieval

Usage

```
ca_scenario(x = ca_apireq(), scenario)
```

Arguments

`x`	Cal-Adapt API request
`scenario`	Abbreviation of emissions scenario(s)

Details

For valid options for `scenario`, see `scenarios`.

---

ca_settings — *Manage package settings*

Description

View and manage package settings

Usage

```
ca_settings(console_colors = NA, date_slice = NA, quiet = FALSE)
```

Arguments

`console_colors`	The name of a preset, or list
`date_slice`	Whether to use date slicing on the Cal-Adapt API when available
`quiet`	Suppress messages, logical

Details

`console_colors` controls the color of text printed at the console. You can pass the name of a preset or a named list of color functions (i.e., from crayon package). Up to six styles are recognized, see example below.

`date_slice` determines whether or not date slicing via URL construction should be used for those Cal-Adapt datasets that support it. This is generally a good idea, but can be set to FALSE for trouble-shooting.

See Also

`ca_getcache`, `ca_setcache`

Examples

```
## Not run:
ca_settings(console_colors = list(ca_accent1 = crayon::blue$bold,
                                  ca_accent2 = crayon::magenta,
                                  ca_accent3 = crayon::red$bold,
                                  ca_accent4 = crayon::red,
                                  ca_message = crayon::silver,
                                  ca_success = crayon::green$bold))

## End(Not run)
```

`ca_slug` *Adds slug(s) to a Cal-Adapt API call*

Description

Specify the raster series slug a Cal-Adapt API call should retrieve

Usage

```
ca_slug(x = ca_apireq(), slug)
```

Arguments

`x`	Cal-Adapt API request
`slug`	Raster series slug(s)

Details

To find valid slugs, see `ca_catalog_rs`.

See Also

`ca_catalog_rs`

`ca_stars_6d` *Create a six-dimensional stars object for modeled climate data*

Description

Create a six-dimensional stars object for modeled climate data

Usage

```
ca_stars_6d(stars_lst, index_tbl = NULL)
```

Arguments

`stars_lst`	A list of stars rasters
`index_tbl`	A tibble of metadata for stars_lst

Details

stars_lst is a list of stars objects downloaded by `ca_getrst_stars` and turned into a list by `ca_stars_read`. Note that both of these functions must use ‘sidecar = TRUE‘.

Creating a six-dimensional stars array of projected climate data may be useful for writing more compact expressions for analysis. Six-dimensional arrays can only be constructed if the API request specifed the GCM, scenario, and climate variable. Rasters retrieved using an API request that specified the dataset by the name of the slug can not be turned into a 6D arrays. Another requirement is that all the rasters have the same location / extent.

Value

A six-dimensional stars object with dimensions x, y, scenario, gcm, date, and cvar

See Also

`ca_getrst_stars`, `ca_stars_read`, `ca_stars_index`

`ca_stars_index` *Create an index for a list of stars rasters*

Description

Create an index for a list of stars rasters

Usage

```
ca_stars_index(x)
```

Arguments

x	A list of stars rasters

Details

When you download rasters from Cal-Adapt using `ca_getrst_stars` `ca_stars_index` generates an index of the properties of the elements of a list of stars rasters to help identify which stars rasters contain which climate model data.

Value

A tibble with the properties of elements of `x`. There will be one row for each element of `x`. Columns include cvar, scenario, gcm, period, slug, livneh, start, end, rows, and cols.

See Also

`ca_getrst_stars`, `ca_stars_read`, `ca_stars_6d`

Examples

```
## Not run:
## Download 5 years of daily max and min temp for Merced County as rasters
mercd_cap <- ca_loc_aoipreset(type = "counties", idfld = "fips", idval = "06047") %>%
  ca_gcm(gcms[1:2]) %>%
  ca_period("day") %>%
  ca_cvar(c("tasmin", "tasmax")) %>%
  ca_scenario("rcp45") %>%
  ca_years(start = 2060, end = 2065)

mercd_stars_lst <- mercd_cap %>%
  ca_getrst_stars(out_dir = ".") %>%
  ca_read_stars()

## Create an index tibble to see the climate model in each stars raster
mercd_stars_tbl <- mercd_stars_lst %>%
  ca_starslist_index()

## End(Not run)
```

`ca_stars_mosaic` *Mosaic stars objects into a seamless array for large areas*

Description

Mosaic stars objects into a seamless array for large areas

Usage

```
ca_stars_mosaic(
  stars_lst,
  index_tbl = NULL,
  geom_mask = NULL,
  combine_6d = FALSE,
  quiet = FALSE
)
```

Arguments

`stars_lst`	A list of stars rasters
`index_tbl`	A tibble of metadata for stars_lst
`geom_mask`	A sf or sfc polygon object to crop the mosaiced raster
`combine_6d`	Combine multiple 3D rasters into one 6D raster
`quiet`	Suppress messages

Details

stars_lst is a list of stars objects downloaded by `ca_getrst_stars` and turned into a list by `ca_stars_read`. Note that both of these functions must be run with ‘sidecar = TRUE‘ (the default).

`combine_6d = TRUE` will return the mosaic as a six-dimensional raster. This further requires that the raster was originally downloaded using an API request that specified the dataset by scenario, GCM, and climate variable (i.e., not by a slug).

Value

If `combine_6d = FALSE`, a list of 3D rasters will be returned (x, y and date/year). If `combine_6d = TRUE`, a 6D stars raster will be returned (x, y, date/year, scenario, GCM, climate variable).

See Also

`ca_getrst_stars`, `ca_stars_read`, `ca_stars_index`

`ca_stars_read` *Read tif files from disk*

Description

Read tif files from disk

Usage

```
ca_stars_read(x, sidecar = TRUE, proxy = FALSE)
```

Arguments

`x`	File name(s) with path of tif files
`sidecar`	Read sidecar files if they exist, logical
`proxy`	Import the TIF file as a stars proxy

Details

This function can be used to read tif files that were downloaded by `ca_getrst_stars`. It is a lightweight wrapper around stars::read_stars(), with the ability to read the sidecar files created by `ca_getrst_stars`. These sidecar files restore all of the attributes of the array dimensions that are not otherwise preserved by the tif format.

`proxy` signifies whether the tif file(s) should be read as stars proxy (e.g., pointer). This is recommended if the rasters are potentially too large to fit into memory. With stars proxy objects, rasters will be read into memory only when needed, at the cost of potentially slightly slower performance. For details see stars documentation.

Value

A list of stars objects

See Also

`ca_getrst_stars`

`ca_years` *Adds the start and end year to a Cal-Adapt API call*

Description

Specifies the start and end year of a Cal-Adapt API call

Usage

```
ca_years(x = ca_apireq(), start, end)
```

Arguments

`x`	Cal-Adapt API request
`start`	start year
`end`	end year

`cvars` *Climate variables*

Description

Climate variables available as raster series

Usage

```
data(cvars)
```

Format

A character vector with the names of climate variables

Details

The following climate variables are available as raster series thru the Cal-Adapt API.

`tasmax`: Maximum Temperature (historical values from UW Hydro and forecast values from LOCA downscaled climate projections)

`tasmin`: Minimum Temperature (historical values from UW Hydro and forecast values from LOCA downscaled climate projections)

`pr`: Precipitation (historical values from UW Hydro and forecast values from LOCA downscaled climate projections)

Source

https://berkeley-gif.github.io/caladapt-docs/data-catalog.html#climate-variables

`format.ca_apireq` *Format a ca_apireq object*

Description

Format a ca_apireq object

Usage

```
## S3 method for class 'ca_apireq'
format(x, ...)
```

Arguments

`x`	Cal-Adapt API request
`...`	Unused

`format.ca_db_info` *Format a ca_db_info object for printing at the console*

Description

Format a ca_db_info object for printing at the console

Usage

```
## S3 method for class 'ca_db_info'
format(x, ...)
```

Arguments

`x`	An object of class ca_db_info
`...`	Unused

`gcms` *Global climate models*

Description

Global Climate Models available through the Cal-Adapt API

Usage

```
data(gcms)
```

Format

An character vector with the names of 10 GCMs (abbreviated)

Details

The following GCMs have been selected by California state agencies as priority models for Fourth Assessment Research and are available thru the Cal-Adapt API. The first four have been identified as the priority GCMs representing (see 4 priority models)

`HadGEM2-ES`: Met Office Hadley Centre and Instituto Nacional de Pesquisas Espaciais

`CNRM-CM5`: Centre National de Recherches Météorologiques/ Centre Européen de Recherche et Formation Avancée en Calcul Scientifique

`CanESM2`: Canadian Centre for Climate Modelling and Analysis

`MIROC5`: Atmosphere and Ocean Research Institute (The University of Tokyo), National Institute for Environmental Studies, and Japan Agency for Marine-Earth Science and Technology

`ACCESS1-0`: Commonwealth Scientific and Industrial Research Organization (CSIRO) and Bureau of Meteorology (BOM) Australia

`CCSM4`: University of Miami - RSMAS

`CESM1-BGC`: Community Earth System Model Contributors

`CMCC-CMS`: Centro Euro-Mediterraneo per I Cambiamenti Climatici

`GFDL-CM3`: NOAA Geophysical Fluid Dynamics Laboratory

`HadGEM2-CC`: Met Office Hadley Centre

Source

https://berkeley-gif.github.io/caladapt-docs/data-catalog.html#global-climate-models-gcm

---

`periods` *Temporal aggregation periods*

Description

Temporal aggregation periods for raster series

Usage

```
data(periods)
```

Format

An character vector with three names of temporal aggregation periods

Details

The following temporal aggregation periods for raster series are available thru the Cal-Adapt API.

`day`: Daily values `month`: Monthly summary statistic `year`: Annual summary statistic

`30yavg`: 30 year summary statistic

Source

https://berkeley-gif.github.io/caladapt-docs/data-catalog.html#period

`plot.ca_apireq` *Plot a ca_apireq object*

Description

Plot a ca_apireq object

Usage

```
## S3 method for class 'ca_apireq'
plot(
  x,
  basemap = c("Esri.NatGeoWorldMap", "OpenStreetMap")[1],
  locagrid = FALSE,
  static = FALSE,
  ...
)
```

Arguments

`x`	Cal-Adapt API request
`basemap`	The name of a basemap tile layer (see tm_basemap)
`locagrid`	Overlay a portion of the LOCA downscaled grid
`static`	Plot a static map instead of a interactive leaflet map
`...`	Unused

`print.ca_apireq` *Print a ca_apireq object*

Description

Print a ca_apireq object

Usage

```
## S3 method for class 'ca_apireq'
print(x, ...)
```

Arguments

`x`	Cal-Adapt API request
`...`	Unused

`print.ca_db_info` *Print a ca_db_info object*

Description

Print a ca_db_info object

Usage

```
## S3 method for class 'ca_db_info'
print(x, ...)
```

Arguments

`x`	An object of class ca_db_info
`...`	Unused

`scenarios` *Scenarios*

Description

Carbon emission scenarios available through the Cal-Adapt API

Usage

```
data(scenarios)
```

Format

An character vector with three abbreviated names of carbon emissions scenarios

Details

The following carbon emission scenarios are available thru the Cal-Adapt API.

`rcp45`: RCP 4.5 (Emissions peak around 2040, then decline)

`rcp85`: RCP 8.5 (Emissions continue to rise strongly through 2050 and plateau)

`historical`: Historical

Source

https://berkeley-gif.github.io/caladapt-docs/data-catalog.html#scenarios

Index

∗ **aoipreset**
    aoipreset_types, 3
∗ **datasets**
    ca_baseurl, 5
    cvars, 32
    gcms, 33
    periods, 34
    scenarios, 36

aoipreset_idflds *(*aoipreset_types*)*, 3
aoipreset_idval *(*aoipreset_types*)*, 3
aoipreset_types, 3, *4*

bbox_resize, 3

ca_aoipreset_geom, 4, *15*
ca_apireq, 5, *21, 22*
ca_baseurl, 5
ca_biggeom_blocks, 6
ca_catalog_fetch, 7, *8, 9*
ca_catalog_rs, *7*, 7, *9*, *15*, *20*, *26, 27*
ca_catalog_search, 8
ca_cvar, 9
ca_dates, 10
ca_db_indices, 10, *18*
ca_db_info, *11*, 11, *13*, *18*
ca_db_read, 12, *18*
ca_example_apireq, 13
ca_gcm, 14
ca_getcache, *4*, *7, 8*, 14, *21*, *27*
ca_getrst_stars, *6*, 15, *19, 20*, *28–31*
ca_getvals_db, *11–13*, *15*, 16, *19, 20*
ca_getvals_tbl, *15*, 19, *21*
ca_livneh, 20
ca_loc_aoipreset, 21
ca_loc_pt, 22, *24*
ca_loc_sf, *22*, 22
ca_locagrid_geom, *15*, 20
ca_options, *21*, *23*, 23
ca_period, 24
ca_preflight, 24
ca_read_stars, *16*
ca_scenario, 26
ca_setcache, *4*, *7*, *27*
ca_setcache *(*ca_getcache*)*, 14
ca_settings, 26
ca_slug, 27
ca_stars_6d, 28, *29*
ca_stars_index, *28*, 28, *30*
ca_stars_mosaic, 29
ca_stars_read, *6*, *28–30*, 30
ca_years, 31
cvars, *9*, 32

filter, *23*
format.ca_apireq, 32
format.ca_db_info, 33

gcms, *14*, 33
geos_unary, *23*

periods, *24*, 34
plot.ca_apireq, 35
print.ca_apireq, 35
print.ca_db_info, 36

scenarios, *26*, 36
slice, *23*
st_cast, *23*