Package 'epwshiftr'

March 12, 2024

Title Create Future 'EnergyPlus' Weather Files using 'CMIP6' Data

Version 0.1.4

Description Query, download climate change projection data from the 'CMIP6' (Coupled Model Intercomparison Project Phase 6) project <https://pcmdi.llnl.gov/CMIP6/> in the 'ESGF' (Earth System Grid Federation) platform <https://esgf.llnl.gov>, and create future 'EnergyPlus' <https://energyplus.net> Weather ('EPW') files adjusted from climate changes using data from Global Climate Models ('GCM').

Imports checkmate (>= 2.0.0), cli (>= 3.4.0), data.table (>= 1.12.4), eplusr (>= 0.15.2), fst, future.apply, jsonlite, PCICt, progressr, psychrolib, R6, rappdirs, RNetCDF, units

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

License MIT + file LICENSE

Encoding UTF-8

URL https://github.com/ideas-lab-nus/epwshiftr

BugReports https://github.com/ideas-lab-nus/epwshiftr/issues

RoxygenNote 7.3.1

Collate 'coord.R' 'dict.R' 'utils.R' 'epwshiftr-package.R' 'esgf.R' 'gh.R' 'morph.R' 'netcdf.R' 'query.R'

VignetteBuilder knitr

Config/testthat/start-first utils, esgf, coord, netcdf, morph

Config/testthat/edition 3

NeedsCompilation no

Author Hongyuan Jia [aut, cre] (<https://orcid.org/0000-0002-0075-8183>), Adrian Chong [aut] (<https://orcid.org/0000-0002-9486-4728>)

Maintainer Hongyuan Jia <hongyuanjia@outlook.com>

Repository CRAN

Date/Publication 2024-03-12 12:40:05 UTC

R topics documented:

`epwshiftr-package` *epwshiftr: Create future EnergyPlus Weather files using CMIP6 data*

Description

Query, download climate change projection data from the CMIP6 (Coupled Model Intercomparison Project Phase 6) project in the ESGF (Earth System Grid Federation) platform, and create future EnergyPlus Weather (EPW) files adjusted from climate changes using data from Global Climate Models (GCM).

Package options

- `epwshiftr.verbose`: If `TRUE`, more detailed message will be printed. Default: `FALSE`.
- `epwshiftr.threshold_alpha`: the threshold of the absolute value for alpha, i.e. monthly-mean fractional change, when performing morphing operations. The default value is set to `3`. If the morphing methods are set `"stretch"` or `"combined"`, and the absolute alpha exceeds the threshold value, warnings are issued and the morphing method fallbacks to `"shift"` to avoid unrealistic morphed values.
- `epwshiftr.dir`: The directory to store package data, including CMIP6 model output file index and etc. If not set, the current user data directory will be used.

Author(s)

Hongyuan Jia

See Also

Useful links:

- https://github.com/ideas-lab-nus/epwshiftr
- Report bugs at https://github.com/ideas-lab-nus/epwshiftr/issues

---

`Cmip6Dict` *CMIP6 Controlled Vocabularies (CVs) and Data Request Dictionary*

Description

The `Cmip6Dict` object provides functionalities to fetch the latest CMIP6 Controlled Vocabularies (CVs) and Data Request (DReq) information.

Usage

```
cmip6_dict()
```

Details

The CMIP6 CVs gives a well-defined set of global attributes that are recorded in each CMIP6 model output, providing information necessary for interpreting the data. The data of CMIP6 CVs is stored as JSON files in the WCRP-CMIP GitHub Repo.

The CMIP6 DReq defines all the quantities from CMIP6 simulations that should be archived. This includes both quantities of general interest needed from most of the CMIP6-endorsed model intercomparison projects (MIPs) and quantities that are more specialized and only of interest to a single endorsed MIP. The raw data of DReq is stored a Microsoft Excel file (`CMIP6_MIP_tables.xlsx`) in a Subversion repo. The `Cmip6Dict` object uses the parsed DReq data that is stored in the GitHub Repo.

For more information, please see:

- CMIP6 Global Attributes, DRS, Filenames, Directory Structure, and CV's
- CMIP6 Data Request

Methods

Public methods:

- `Cmip6Dict$version()`
- `Cmip6Dict$is_empty()`
- `Cmip6Dict$timestamp()`
- `Cmip6Dict$built_time()`
- `Cmip6Dict$build()`
- `Cmip6Dict$get()`
- `Cmip6Dict$save()`
- `Cmip6Dict$load()`

- `Cmip6Dict$print()`

Method `version()`**:** Get the version of CVs and Data Request

Usage:

```
Cmip6Dict$version()
```

Returns: A list of two element:

- `cvs`: A numeric_version object giving the version of CVs
- `dreq`: A numeric_version object giving the version of Data Request

Method `is_empty()`**:** Is it an empty Cmip6Dict?
`$is_empty()` checks if this `Cmip6Dict` is empty, i.e. the `$build()` or `$load()` method hasn't been called yet and there is no data of CVs and Data Request.

Usage:

```
Cmip6Dict$is_empty()
```

Returns: A single logical value of `TRUE` or `FALSE`.

Method `timestamp()`**:** Get the last modified time for CVs

Usage:

```
Cmip6Dict$timestamp()
```

Returns: A list of 14 DateTimes:

- `"cvs"`: The last modified time for the whole CV collection
- `"drs"`: The last modified time for Data Reference Syntax (DRS)
- `"activity_id"`: The last modified time for Activity ID
- `"experiment_id"`: The last modified time for Experiment ID
- `"frequency"`: The last modified time for Frequency
- `"grid_label"`: The last modified time for Grid Label
- `"institution_id"`: The last modified time for Institution ID
- `"nominal_resolution"`: The last modified time for Nominal Resolution
- `"realm"`: The last modified time for Realm
- `"required_global_attributes"`: The last modified time for Required Global Attributes
- `"source_id"`: The last modified time for Source ID
- `"source_type"`: The last modified time for Source Type
- `"sub_experiment_id"`: The last modified time for Sub-Experiment ID
- `"table_id"`: The last modified time for Table ID

Method `built_time()`**:** Get the time when the dictionary was built

Usage:

```
Cmip6Dict$built_time()
```

Returns: A DateTime

Method `build()`**:** Fetch and parse all data of CVs and Data Request

Usage:

```
Cmip6Dict$build(token = NULL, force = FALSE)
```

Arguments:

`token` A string of GitHub token that is used to access GitHub REST APIs. If `NULL`, `GITHUB_PAT` or `GITHUB_TOKEN` environment variable will be used if exists. Default: `NULL`.

`force` Whether to force to rebuild the dict when it has been already built before. Default: `FALSE`.

Returns: The updated `Cmip6Dict` itself.

Method `get()`**:** Get the data for a specific CV or Data Request

Usage:

```
Cmip6Dict$get(type)
```

Arguments:

`type` A single string indicating the type of data to list. Should be one of:

- `"drs"`: Data Reference Syntax (DRS)
- `"activity_id"`: Activity ID
- `"experiment_id"`: Experiment ID
- `"frequency"`: Frequency
- `"grid_label"`: Grid Label
- `"institution_id"`: Institution ID
- `"nominal_resolution"`: Nominal Resolution
- `"realm"`: Realm
- `"required_global_attributes"`: Required Global Attributes
- `"source_id"`: Source ID
- `"source_type"`: Source Type
- `"sub_experiment_id"`: Sub-Experiment ID
- `"table_id"`: Table ID
- `"dreq"`: Data Request

Returns: For `"drs"`, "activity_id", "frequency", "grid_label", "institution_id", "source_type"and"sub_experiment_id' a list.
For `"experiment_id"`, `"source_id"` and `"dreq"`, a data.table.
For `"nominal_resolution"`, `"required_global_attributes"` and `"table_id"`, a character vector.

Method `save()`**:** Save the Cmip6Dict object

`$save()` stores all the core data of current `Cmip6Dict` object into an RDS file named `CMIP6DICT` in the specified folder. This file can be reloaded via `$load()` method to restore the last state of current `Cmip6Dict` object.

Usage:

```
Cmip6Dict$save(dir = getOption("epwshiftr.dir", "."))
```

Arguments:

`dir` A single string giving the directory to save the RDS file. Default is set to the global option `epwshiftr.dir`. The directory will be created if not exists. If this global option is not set, the current working directory is used.

Returns: A single string giving the full path of the RDS file.

Method `load()`**:** Load the saved Cmip6Dict object from file
`$load()` loads the RDS file named `CMIP6DICT` that is created using `$save()` method.
Please note that the file should be exactly the same as `CMIP6DICT` without file extension.

Usage:

```
Cmip6Dict$load(dir = getOption("epwshiftr.dir", "."))
```

Arguments:

`dir` A single string giving the directory to find the RDS file. Default is set to the global option `epwshiftr.dir`. If this global option is not set, the current working directory is used.

Returns: A single string giving the full path of the RDS file.

Method `print()`**:** Print a summary of the current `Cmip6Dict` object
`$print()` gives the summary of current `Cmip6Dict` object including the version of CVs and Data Request, and the last built time.

Usage:

```
Cmip6Dict$print()
```

Returns: The `Cmip6Dict` object itself, invisibly.

Author(s)

Hongyuan Jia

Examples

```
## Not run:

# create a new Cmip6Dict object
dict <- cmip6_dict()

# by default, there is no data when the Cmip6Dict was created
dict$is_empty()

# fetch and parse all CVs and Data Request data
dict$build()

# get the version of CVs nand Data Request
dict$version()

# get the last modified time for each CV and Data Request
dict$timestamp()

# get the time when the dict was built
dict$built_time()

# get the data of CVs and DReq
dict$get("activity_id")
dict$get("experiment_id")
dict$get("sub_experiment_id")
dict$get("institution_id")
dict$get("source_id")
```

```
dict$get("table_id")
dict$get("frequency")
dict$get("grid_label")
dict$get("realm")
dict$get("source_type")
dict$get("dreq")

# save the dict object for later usage
# default location is the value of global option "epwshiftr.dir"
dict$save()

# the saved dict object can be reloaded
new_dict <- cmip6_dict()
new_dict$load()

# print will show the version summary and the last built time
dict$print()

## End(Not run)
```

`EsgfQuery` *Query CMIP6 data using ESGF search RESTful API*

Description

The Earth System Grid Federation (ESGF) is an international collaboration for the software that powers most global climate change research, notably assessments by the Intergovernmental Panel on Climate Change (IPCC).

The ESGF search service exposes RESTful APIs that can be used by clients to query the contents of the underlying search index, and return results matching the given constraints. The documentation of the APIs can be found using this link

`EsgfQuery` is the workhorse for dealing with ESGF search services.

Usage

```
query_esgf(host = "https://esgf-node.llnl.gov/esg-search")
```

Arguments

`host`	The URL to the ESGF Search API service. This should be the URL of the ESGF search service excluding the final endpoint name. Usually this is `http://<hostname>/esg-search`. Default is set to the LLNL (Lawrence Livermore National Laboratory) Index Node, which is `"https://esgf-node.llnl.gov/esg-search"`.

`EsgfQuery` object

`query_esgf()` returns an `EsgfQuery` object, which is an R6 object with quite a few methods that can be classified into 3 categories:

- **Value listing**: methods to list all possible values of facets, shards, etc.
- **Parameter getter & setter**: methods to get the query parameter values or set them before sending the actual query to the ESGF search services.
- **Query responses**: methods to collect results for the query response.

Value listing:

When creating an `EsgfQuery` object, a facet listing query is sent to the index node to get all available facets and shards for the default project (CMIP6). `EsgfQuery` object provides three value-listing methods to extract data from the response of the facet listing query:

- `EsgfQuery$list_all_facets()`: List all available facet names.
- `EsgfQuery$list_all_shards()`: List all available shards.
- `EsgfQuery$list_all_values()`: List all available values of a specific facet.

Parameter getter & setter:

The ESGF search services support a lot of parameters. The `EsgfQuery` contains dedicated methods to set values for most of them, including:

- Most common keywords: `facets`, `offset`, `limit`, `fields`, `type`, `replica`, `latest`, `distrib` and `shards`.
- Most common facets: `project`, `activity_id`, `experiment_id`, `source_id`, `variable_id`, `frequency`, `variant_label`, `nominal_resolution` and `data_node`.

All methods act in a similar way:

- If input is given, the corresponding parameter is set and the updated `EsgfQuery` object is returned.
  - This makes it possible to chain different parameter setters, e.g. `EsgfQuery$project("CMIP6")$frequency("day` sets the parameter `project`, `frequency` and `limit` sequentially.
  - For parameters that want character inputs, you can put a preceding `!` to negate the constraints, e.g. `EsgfQuery$project(!"CMIP6")` searches for all projects except for `CMIP6`.
- If no input is given, the current parameter value is returned. For example, directly calling `EsgfQuery$project()` returns the current value of the `project` parameter. The returned value can be two types:
  - `NULL`, i.e. there is no constraint on the corresponding parameter
  - An `EsgfQueryParam` object which is essentially a list of three elements:
    - `value`: The input values
    - `negate`: Whether there is a preceding `!` in the input
    - `name`: The parameter name

Despite methods for specific keywords and facets, you can specify arbitrary query parameters using `EsgfQuery$params()` method. For details on the usage, please see the documentation.

Query responses:

The query is not sent unless related methods are called:

- `EsgfQuery$count()`: Count the total number of records that match the query.
  - You can return only the total number of matched record by calling `EsgfQuery$count(facets = FALSE)`
  - You can also count the matched records for specified facets, e.g. `EsgfQuery$count(facets = c("source_id", "activity_id"))`
- `EsgfQuery$collect()`: Collect the query results and format it into a data.table

Other helpers:

`EsgfQuery` object also provide several other helper functions:

- `EsgfQuery$build_cache()`: By default, `EsgfQuery$build_cache()` is called when initialize a new `EsgfQuery` object. So in general, there is no need to call this separately. Basically, `EsgfQuery$build_cahce()` sends a facet listing query to the index node and stores the response internally. The response contains all available facets and shards and is used as a source for validating user input for parameter setters.
- `EsgfQuery$url()`: Returns the actual query URL or the wget script URL which can be used to download all files matching the given constraints..
- `EsgfQuery$response()`: Returns the actual response of `EsgfQuery$count()` and `EsgfQuery$collect()`. It is a named list generated from the JSON response using `jsonlite::fromJSON()`.
- `EsgfQuery$print()`: Print a summary of the current `EsgfQuery` object including the host URL, the built time of facet cache and all query parameters.

Methods

Public methods:

- `EsgfQuery$new()`
- `EsgfQuery$build_cache()`
- `EsgfQuery$list_all_facets()`
- `EsgfQuery$list_all_shards()`
- `EsgfQuery$list_all_values()`
- `EsgfQuery$project()`
- `EsgfQuery$activity_id()`
- `EsgfQuery$experiment_id()`
- `EsgfQuery$source_id()`
- `EsgfQuery$variable_id()`
- `EsgfQuery$frequency()`
- `EsgfQuery$variant_label()`
- `EsgfQuery$nominal_resolution()`
- `EsgfQuery$data_node()`
- `EsgfQuery$facets()`
- `EsgfQuery$fields()`
- `EsgfQuery$shards()`
- `EsgfQuery$replica()`
- `EsgfQuery$latest()`
- `EsgfQuery$type()`

- EsgfQuery$limit()
- EsgfQuery$offset()
- EsgfQuery$distrib()
- EsgfQuery$params()
- EsgfQuery$url()
- EsgfQuery$count()
- EsgfQuery$collect()
- EsgfQuery$response()
- EsgfQuery$print()

Method `new()`**:** Create a new EsgfQuery object

When initialization, a facet listing query is sent to the index node to get all available facets and shards. This information will be used to validate inputs for `activity_id`, `scource_id` facets and etc.

Usage:

```
EsgfQuery$new(host = "https://esgf-node.llnl.gov/esg-search")
```

Arguments:

`host` The URL to the ESGF Search API service. This should be the URL of the ESGF search service excluding the final endpoint name. Usually this is `http://<hostname>/esg-search`. Default is to ses the LLNL (Lawrence Livermore National Laboratory) Index Node, which is `"https://esgf-node.llnl.gov/esg-search"`.

Returns: An `EsgfQuery` object.

Examples:

```
\dontrun{
q <- EsgfQuery$new(host = "https://esgf-node.llnl.gov/esg-search")
q
}
```

Method `build_cache()`**:** Build facet cache used for input validation

A facet cache is data that is fetched using a facet listing query to the index node. It contains all available facets and shards that can be used as parameter values within a specific project.

By default, `$build_cache()` is called when initialize a new `EsgfQuery` object for the default project (CMIP6). So in general, there is no need to call this method, unless that you want to rebuild the cache again with different projects after calling `$project()`.

Usage:

```
EsgfQuery$build_cache()
```

Returns: The modified `EsgfQuery` object.

Examples:

```
\dontrun{
q$build_cache()
}
```

Method `list_all_facets()`**:** List all available facet names

Usage:

```
EsgfQuery$list_all_facets()
```

Returns: A character vector.

Examples:

```
\dontrun{
q$list_all_facets()
}
```

Method `list_all_shards()`**:** List all available shards

Usage:

```
EsgfQuery$list_all_shards()
```

Returns: A character vector.

Examples:

```
\dontrun{
q$list_all_shards()
}
```

Method `list_all_values()`**:** List all available values of a specific facet

Usage:

```
EsgfQuery$list_all_values(facet)
```

Arguments:

`facet` A single string giving the facet name.

Returns: A named character vector.

Examples:

```
\dontrun{
q$list_all_values()
}
```

Method `project()`**:** Get or set the `project` facet parameter.

Usage:

```
EsgfQuery$project(value = "CMIP6")
```

Arguments:

`value` The parameter value. Default: `"CMIP6"`. There are two options:

- If `value` is not given, current value is returned.
- A character vector or `NULL`. Note that you can put a preceding `!` to negate the facet constraints. For example, `$project(!c("CMIP5", "CMIP6"))` searches for all `projects` except for `CMIP5` and `CMIP6`.

Returns:

- If `value` is given, the modified `EsgfQuery` object.
- Otherwise, an `EsgfQueryParam` object which is essentially a list of three elements:
  - `value`: input values.
  - `negate`: Whether there is a preceding `!`.
  - `name`: Parameter name.

Examples:

```
\dontrun{
# get current value
q$project()

# set the parameter
q$project("CMIP6")

# negate the project constraints
q$project(!"CMIP6")

# remove the parameter
q$project(NULL)
}
```

Method `activity_id()`**:** Get or set the `activity_id` facet parameter.

Usage:

```
EsgfQuery$activity_id(value)
```

Arguments:

`value` The parameter value. Default: `NULL`. There are two options:

- If `value` is not given, current value is returned.
- A character vector or `NULL`. Note that you can put a preceding `!` to negate the facet constraints. For example, `$activity_id(!c("C4MIP", "GeoMIP"))` searches for all `activity_id`s except for `C4MIP` and `GeoMIP`.

Returns:

- If `value` is given, the modified `EsgfQuery` object.
- Otherwise, an `EsgfQueryParam` object which is essentially a list of three elements:
  - `value`: input values.
  - `negate`: Whether there is a preceding `!`.
  - `name`: Parameter name.

Examples:

```
\dontrun{
# get current value
q$activity_id()

# set the parameter
q$activity_id("ScenarioMIP")

# negate the constraints
q$activity_id(!c("CFMIP", "ScenarioMIP"))

# remove the parameter
q$activity_id(NULL)
}
```

Method `experiment_id()`**:** Get or set the `experiment_id` facet parameter.

Usage:

```
EsgfQuery$experiment_id(value)
```

Arguments:

`value` The parameter value. Default: `NULL`. There are two options:

- If `value` is not given, current value is returned.
- A character vector or `NULL`. Note that you can put a preceding `!` to negate the facet constraints. For example, `$experiment_id(!c("ssp126", "ssp245"))` searches for all `experiment_id`s except for `ssp126` and `ssp245`.

Returns:

- If `value` is given, the modified `EsgfQuery` object.
- Otherwise, an `EsgfQueryParam` object which is essentially a list of three elements:
  - `value`: input values.
  - `negate`: Whether there is a preceding `!`.
  - `name`: Parameter name.

Examples:

```
\dontrun{
# get current value
q$experiment_id()

# set the parameter
q$experiment_id(c("ssp126", "ssp585"))

# negate the constraints
q$experiment_id(!c("ssp126", "ssp585"))

# remove the parameter
q$experiment_id(NULL)
}
```

Method `source_id()`**:** Get or set the `source_id` facet parameter.

Usage:

```
EsgfQuery$source_id(value)
```

Arguments:

`value` The parameter value. Default: `NULL`. There are two options:

- If `value` is not given, current value is returned.
- A character vector or `NULL`. Note that you can put a preceding `!` to negate the facet constraints. For example, `$source_id(!c("CESM2", "CESM2-FV2"))` searches for all `source_id`s except for `CESM2` and `CESM2-FV2`.

Returns:

- If `value` is given, the modified `EsgfQuery` object.
- Otherwise, an `EsgfQueryParam` object which is essentially a list of three elements:
  - `value`: input values.
  - `negate`: Whether there is a preceding `!`.

- name: Parameter name.

Examples:

```
\dontrun{
# get current value
q$source_id()

# set the parameter
q$source_id(c("BCC-CSM2-MR", "CESM2"))

# negate the constraints
q$source_id(!c("BCC-CSM2-MR", "CESM2"))

# remove the parameter
q$source_id(NULL)
}
```

Method `variable_id()`**:** Get or set the `variable_id` facet parameter.

Usage:

```
EsgfQuery$variable_id(value)
```

Arguments:

`value` The parameter value. Default: `NULL`. There are two options:

- If `value` is not given, current value is returned.
- A character vector or `NULL`. Note that you can put a preceding `!` to negate the facet constraints. For example, `$variable_id(!c("tas", "pr"))` searches for all `variable_ids` except for `tas` and `pr`.

Returns:

- If `value` is given, the modified `EsgfQuery` object.
- Otherwise, an `EsgfQueryParam` object which is essentially a list of three elements:
  - `value`: input values.
  - `negate`: Whether there is a preceding `!`.
  - `name`: Parameter name.

Examples:

```
\dontrun{
# get current value
q$variable_id()

# set the parameter
q$variable_id(c("tas", "pr"))

# negate the constraints
q$variable_id(!c("tas", "pr"))

# remove the parameter
q$variable_id(NULL)
}
```

Method `frequency()`**:** Get or set the `frequency` facet parameter.

Usage:

```
EsgfQuery$frequency(value)
```

Arguments:

`value` The parameter value. Default: `NULL`. There are two options:

- If `value` is not given, current value is returned.
- A character vector or `NULL`. Note that you can put a preceding `!` to negate the facet constraints. For example, `$frequency(!c("day", "mon"))` searches for all `frequency`s except for `day` and `mon`.

Returns:

- If `value` is given, the modified `EsgfQuery` object.
- Otherwise, an `EsgfQueryParam` object which is essentially a list of three elements:
  - `value`: input values.
  - `negate`: Whether there is a preceding `!`.
  - `name`: Parameter name.

Examples:

```
\dontrun{
# get current value
q$frequency()

# set the parameter
q$frequency(c("1hr", "day"))

# negate the constraints
q$frequency(!c("1hr", "day"))

# remove the parameter
q$frequency(NULL)
}
```

Method `variant_label()`**:** Get or set the `variant_label` facet parameter.

Usage:

```
EsgfQuery$variant_label(value)
```

Arguments:

`value` The parameter value. Default: `NULL`. There are two options:

- If `value` is not given, current value is returned.
- A character vector or `NULL`. Note that you can put a preceding `!` to negate the facet constraints. For example, `$variant_label(!c("r1i1p1f1", "r2i1p1f1"))` searches for all `variant_label`s except for `r1i1p1f1` and `r2i1p1f1`.

Returns:

- If `value` is given, the modified `EsgfQuery` object.
- Otherwise, an `EsgfQueryParam` object which is essentially a list of three elements:
  - `value`: input values.

- negate: Whether there is a preceding !.
- name: Parameter name.

Examples:

```
\dontrun{
# get current value
q$variant_label()

# set the parameter
q$variant_label(c("r1i1p1f1", "r1i2p1f1"))

# negate the constraints
q$variant_label(!c("r1i1p1f1", "r1i2p1f1"))

# remove the parameter
q$variant_label(NULL)
}
```

Method `nominal_resolution()`**:** Get or set the `nominal_resolution` facet parameter.

Usage:

```
EsgfQuery$nominal_resolution(value)
```

Arguments:

`value` The parameter value. Default: `NULL`. There are two options:

- If `value` is not given, current value is returned.
- A character vector or `NULL`. Note that you can put a preceding `!` to negate the facet constraints. For example, `$nominal_resolution(!c("50 km", "1x1 degree"))` searches for all `nominal_resolutions` except for `50 km` and `1x1 degree`.

Returns:

- If `value` is given, the modified `EsgfQuery` object.
- Otherwise, an `EsgfQueryParam` object which is essentially a list of three elements:
  - `value`: input values.
  - `negate`: Whether there is a preceding `!`.
  - `name`: Parameter name.

Examples:

```
\dontrun{
# get current value
q$nominal_resolution()

# set the parameter
q$nominal_resolution(c("100 km", "1x1 degree"))

# negate the constraints
q$nominal_resolution(!c("100 km", "1x1 degree"))

# remove the parameter
q$nominal_resolution(NULL)
}
```

Method `data_node()`**:** Get or set the `data_node` parameter.

Usage:

```
EsgfQuery$data_node(value)
```

Arguments:

`value` The parameter value. Default: `NULL`. There are two options:

- If `value` is not given, current value is returned.
- A character vector or `NULL`. Note that you can put a preceding `!` to negate the facet constraints. For example, `$data_node(!c("cmip.bcc.cma.cn", "esg.camscma.cn"))` searches for all `data_nodes` except for `cmip.bcc.cma.cn` and `esg.camscma.cn`.

Returns:

- If `value` is given, the modified `EsgfQuery` object.
- Otherwise, an `EsgfQueryParam` object which is essentially a list of three elements:
  - `value`: input values.
  - `negate`: Whether there is a preceding `!`.
  - `name`: Parameter name.

Examples:

```
\dontrun{
# get current value
q$data_node()

# set the parameter
q$data_node("esg.lasg.ac.cn")

# negate the constraints
q$data_node(!"esg.lasg.ac.cn")

# remove the parameter
q$data_node(NULL)
}
```

Method `facets()`**:** Get or set the `facets` parameter for facet counting query.
Note that `$facets()` only affects `$count()` method when sending a query of facet counting.

Usage:

```
EsgfQuery$facets(value)
```

Arguments:

`value` The facet parameter value. Default: `NULL`. There are two options:

- If `value` is not given, current value is returned.
- A character vector or `NULL`. The special notation `"*"` can be used to indicate that all available facets should be considered.

Returns:

- If `value` is given, the modified `EsgfQuery` object.
- Otherwise, an `EsgfQueryParam` object which is essentially a list of three elements:
  - `value`: input values.

- negate: Whether there is a preceding !.
- name: Parameter name.

Examples:

```
\dontrun{
# get current value
q$facets()

# set the facets
q$facets(c("activity_id", "source_id"))

# use all available facets
q$facets("*")
}
```

Method `fields()`**:** Get or set the `fields` parameter.

By default, all available metadata fields are returned for each query. `$facets()` can be used to limit the number of fields returned in the query response.

Usage:

```
EsgfQuery$fields(value = "*")
```

Arguments:

`value` The facet parameter value. Default: `"*"`. There are two options:

- If `value` is not given, current value is returned.
- A character vector or `NULL`. The special notation `"*"` can be used to indicate that all available fields should be considered.

Returns:

- If `value` is given, the modified `EsgfQuery` object.
- Otherwise, an `EsgfQueryParam` object which is essentially a list of three elements:
  - `value`: input values.
  - `negate`: Whether there is a preceding !.
  - `name`: Parameter name.

Examples:

```
\dontrun{
# get current value
q$fields()

# set the fields
q$fields(c("activity_id", "source_id"))

# use all available fields
q$fields("*")

# remove the parameter
# act the same as above because the default `fields` in ESGF search
# services is `*` if `fields` is not specified
q$fields(NULL)
}
```

Method `shards()`**:** Get or set the `shards` parameter.

By default, a distributed query targets all ESGF Nodes. `$shards()` can be used to execute a distributed search that targets only one or more specific nodes.

All available shards can be retrieved using `$list_all_shards()` method.

Usage:

```
EsgfQuery$shards(value)
```

Arguments:

`value` The facet parameter value. There are two options:

- If `value` is not given, current value is returned.
- A character vector or `NULL`.

Returns:

- If `value` is given, the modified `EsgfQuery` object.
- Otherwise, an `EsgfQueryParam` object which is essentially a list of three elements:
  - `value`: input values.
  - `negate`: Whether there is a preceding `!`.
  - `name`: Parameter name.

Examples:

```
\dontrun{
# get current value
q$shards()

# set the parameter
q$shards("localhost:8983/solr/datasets")

# negate the constraints
q$shards(!"localhost:8983/solr/datasets")

# only applicable for distributed queries
q$distrib(FALSE)$shards("localhost:8983/solr/datasets") # Error

# remove the parameter
q$shards(NULL)
}
```

Method `replica()`**:** Get or set the `replica` parameter.

By default, a query returns all records (masters and replicas) matching the search criteria, i.e. `$replica(NULL)`. To return only master records, use `$replica(FALSE)`; to return only replicas, use `$replica(TRUE)`.

Usage:

```
EsgfQuery$replica(value)
```

Arguments:

`value` The facet parameter value. Default: `NULL`. There are two options:

- If `value` is not given, current value is returned.
- A flag or `NULL`.

Returns:

- If value is given, the modified EsgfQuery object.
- Otherwise, an EsgfQueryParam object which is essentially a list of three elements:
  - value: input values.
  - negate: Whether there is a preceding !.
  - name: Parameter name.

Examples:

```
\dontrun{
# get current value
q$replica()

# set the parameter
q$replica(TRUE)

# remove the parameter
q$replica(NULL)
}
```

Method `latest()`**:** Get or set the `latest` parameter.

By default, a query to the ESGF search services returns only the very last, up-to-date version of the matching records, i.e. `$latest(TRUE)`. You can use `$latest(FALSE)` to return all versions.

Usage:

```
EsgfQuery$latest(value = TRUE)
```

Arguments:

`value` The facet parameter value. Default: `TRUE`. There are two options:

- If value is not given, current value is returned.
- A flag.

Returns:

- If value is given, the modified EsgfQuery object.
- Otherwise, an EsgfQueryParam object which is essentially a list of three elements:
  - value: input values.
  - negate: Whether there is a preceding !.
  - name: Parameter name.

Examples:

```
\dontrun{
# get current value
q$latest()

# set the parameter
q$latest(TRUE)
}
```

Method `type()`**:** Get or set the type parameter.

There are three types in total: `Dataset`, `File` or `Aggregation`.

Usage:

```
EsgfQuery$type(value = "Dataset")
```

Arguments:

`value` The facet parameter value. Default: `"Dataset"`. There are two options:

- If `value` is not given, current value is returned.
- A string.

Returns:

- If `value` is given, the modified `EsgfQuery` object.
- Otherwise, an `EsgfQueryParam` object which is essentially a list of three elements:
  - `value`: input values.
  - `negate`: Whether there is a preceding `!`.
  - `name`: Parameter name.

Examples:

```
\dontrun{
# get current value
q$type()

# set the parameter
q$type("Dataset")
}
```

Method `limit()`**:** Get or set the `limit` parameter.
`$limit()` can be used to limit the number of records to return. Note that the maximum number of records to return per query for ESGF search services is 10,000. A warning is issued if input value is greater than that. In this case, `limit` will be reset to 10,000.

Usage:

```
EsgfQuery$limit(value = 10L)
```

Arguments:

`value` The facet parameter value. Default: `10`. There are two options:

- If `value` is not given, current value is returned.
- An integer.

Returns:

- If `value` is given, the modified `EsgfQuery` object.
- Otherwise, an `EsgfQueryParam` object which is essentially a list of three elements:
  - `value`: input values.
  - `negate`: Whether there is a preceding `!`.
  - `name`: Parameter name.

Examples:

```
\dontrun{
# get current value
q$limit()

# set the parameter
```

```
q$limit(10L)

# `limit` is reset to 10,000 if input is greater than that
q$limit(10000L) # warning
}
```

Method `offset()`**:** Get or set the `offset` parameter.
If the query returns records that exceed the `limit` number, `$offset()` can be used to paginate through the available results.

Usage:

```
EsgfQuery$offset(value = 0L)
```

Arguments:

`value` The facet parameter value. Default: `0`. There are two options:

- If `value` is not given, current value is returned.
- An integer.

Returns:

- If `value` is given, the modified `EsgfQuery` object.
- Otherwise, an `EsgfQueryParam` object which is essentially a list of three elements:
  - `value`: input values.
  - `negate`: Whether there is a preceding `!`.
  - `name`: Parameter name.

Examples:

```
\dontrun{
# get current value
q$offset()

# set the parameter
q$offset(0L)
}
```

Method `distrib()`**:** Get or set the `distrib` facet
By default, the query is sent to all ESGF Nodes, i.e. `$distrib(TRUE)`. `$distrib(FALSE)` can be used to execute the query only on the target node.

Usage:

```
EsgfQuery$distrib(value = TRUE)
```

Arguments:

`value` The facet parameter value. Default: `TRUE`. There are two options:

- If `value` is not given, current value is returned.
- A flag.

Returns:

- If `value` is given, the modified `EsgfQuery` object.
- Otherwise, an `EsgfQueryParam` object which is essentially a list of three elements:
  - `value`: input values.

- negate: Whether there is a preceding !.
- name: Parameter name.

Examples:

```
\dontrun{
# get current value
q$distrib()

# set the parameter
q$distrib(TRUE)
}
```

Method `params()`**:** Get or set other parameters.
`$params()` can be used to specify other parameters that do not have a dedicated method, e.g. `version`, `master_id`, etc. It can also be used to overwrite existing parameter values specified using methods like `$activity_id()`.

Usage:

```
EsgfQuery$params(...)
```

Arguments:

`...` Parameter values to set. There are three options:

- If not given, existing parameters that do not have a dedicated method are returned.
- If `NULL`, all existing parameters that do not have a dedicated method are removed.
- A named vector, e.g. `$params(score = 1, table_id = "day")` will set `score` to `1` and `table_id` to `day`. The `!` notation can still be used to negate the constraints, e.g. `$params(table_id = !c("3hr", "day"))` searches for all `table_id` except for `3hr` and `day`.

Returns:

- If parameters are specified, the modified `EsgfQuery` object, invisibly.
- Otherwise, an empty list for `$params(NULL)` or a list of `EsgfQueryParam` objects.

Examples:

```
\dontrun{
# get current values
# default is an empty list (`list()`)
q$params()

# set the parameter
q$params(table_id = c("3hr", "day"), member_id = "00")
q$params()

# reset existing parameters
q$frequency("day")
q$params(frequency = "mon")
q$frequency() # frequency value has been changed using $params()

# negating the constraints is also supported
q$params(table_id = !c("3hr", "day"))
```

```
# use NULL to remove all parameters
q$params(NULL)$params()
}
```

Method `url()`**:** Get the URL of actual query or wget script

Usage:

```
EsgfQuery$url(wget = FALSE)
```

Arguments:

`wget` Whether to return the URL of the wget script that can be used to download all files matching the given constraints. Default: `FALSE`.

Returns: A single string.

Examples:

```
\dontrun{
q$url()

# get the wget script URL
q$url(wget = TRUE)

# You can download the wget script using the URL directly. For
# example, the code below downloads the script and save it as
# 'wget.sh' in R's temporary folder:
download.file(q$url(TRUE), file.path(tempdir(), "wget.sh"), mode = "wb")

}
```

Method `count()`**:** Send a query of facet counting and fetch the results

Usage:

```
EsgfQuery$count(facets = TRUE)
```

Arguments:

`facets` `NULL`, a flag or a character vector. There are three options:

- If `NULL` or `FALSE`, only the total number of matched records is returned.
- If `TRUE`, the value of `$facets()` is used to limit the facets. This is the default value.
- If a character vector, it is used to limit the facets.

Returns:

- If `facets` equals `NULL` or `FALSE`, or `$facets()` returns `NULL`, an integer.
- Otherwise, a named list with the first element always being `total` which is the total number of matched records. Other elements have the same length as input facets and are all named integer vectors.

Examples:

```
\dontrun{
# get the total number of matched records
q$count(NULL) # or q$count(facets = FALSE)
```

```
# count records for specific facets
q$facets(c("activity_id", "source_id"))$count()

# same as above
q$count(facets = c("activity_id", "source_id"))
}
```

Method `collect()`**:** Send the actual query and fetch the results
`$collect()` sends the actual query to the ESGF search services and returns the results in a data.table::data.table. The columns depend on the value of query type and `fields` parameter.

Usage:

```
EsgfQuery$collect()
```

Returns: A data.table.

Examples:

```
\dontrun{
q$fields("source_id")
q$collect()
}
```

Method `response()`**:** Get the response of last sent query
The response of the last sent query is always stored internally and can be retrieved using `$response()`. It is a named list generated from the JSON response using `jsonlite::fromJSON()`.

Usage:

```
EsgfQuery$response()
```

Returns: A named list.

Examples:

```
\dontrun{
q$response()
}
```

Method `print()`**:** Print a summary of the current `EsgfQuery` object
`$print()` gives the summary of current `EsgfQuery` object including the host URL, the built time of facet cache and all query parameters.

Usage:

```
EsgfQuery$print()
```

Returns: The `EsgfQuery` object itself, invisibly.

Examples:

```
\dontrun{
q$print()
}
```

Author(s)

Hongyuan Jia

Examples

```
## ------------------------------------------------
## Method `EsgfQuery$new`
## ------------------------------------------------

## Not run:
q <- EsgfQuery$new(host = "https://esgf-node.llnl.gov/esg-search")
q

## End(Not run)

## ------------------------------------------------
## Method `EsgfQuery$build_cache`
## ------------------------------------------------

## Not run:
q$build_cache()

## End(Not run)

## ------------------------------------------------
## Method `EsgfQuery$list_all_facets`
## ------------------------------------------------

## Not run:
q$list_all_facets()

## End(Not run)

## ------------------------------------------------
## Method `EsgfQuery$list_all_shards`
## ------------------------------------------------

## Not run:
q$list_all_shards()

## End(Not run)

## ------------------------------------------------
## Method `EsgfQuery$list_all_values`
## ------------------------------------------------

## Not run:
q$list_all_values()

## End(Not run)

## ------------------------------------------------
## Method `EsgfQuery$project`
## ------------------------------------------------
```

```
## Not run:
# get current value
q$project()

# set the parameter
q$project("CMIP6")

# negate the project constraints
q$project(!"CMIP6")

# remove the parameter
q$project(NULL)

## End(Not run)

## ------------------------------------------------
## Method `EsgfQuery$activity_id`
## ------------------------------------------------

## Not run:
# get current value
q$activity_id()

# set the parameter
q$activity_id("ScenarioMIP")

# negate the constraints
q$activity_id(!c("CFMIP", "ScenarioMIP"))

# remove the parameter
q$activity_id(NULL)

## End(Not run)

## ------------------------------------------------
## Method `EsgfQuery$experiment_id`
## ------------------------------------------------

## Not run:
# get current value
q$experiment_id()

# set the parameter
q$experiment_id(c("ssp126", "ssp585"))

# negate the constraints
q$experiment_id(!c("ssp126", "ssp585"))

# remove the parameter
q$experiment_id(NULL)

## End(Not run)
```

```
## ------------------------------------------------
## Method `EsgfQuery$source_id`
## ------------------------------------------------

## Not run:
# get current value
q$source_id()

# set the parameter
q$source_id(c("BCC-CSM2-MR", "CESM2"))

# negate the constraints
q$source_id(!c("BCC-CSM2-MR", "CESM2"))

# remove the parameter
q$source_id(NULL)

## End(Not run)

## ------------------------------------------------
## Method `EsgfQuery$variable_id`
## ------------------------------------------------

## Not run:
# get current value
q$variable_id()

# set the parameter
q$variable_id(c("tas", "pr"))

# negate the constraints
q$variable_id(!c("tas", "pr"))

# remove the parameter
q$variable_id(NULL)

## End(Not run)

## ------------------------------------------------
## Method `EsgfQuery$frequency`
## ------------------------------------------------

## Not run:
# get current value
q$frequency()

# set the parameter
q$frequency(c("1hr", "day"))

# negate the constraints
q$frequency(!c("1hr", "day"))

# remove the parameter
```

```
q$frequency(NULL)

## End(Not run)

## ------------------------------------------------
## Method `EsgfQuery$variant_label`
## ------------------------------------------------

## Not run:
# get current value
q$variant_label()

# set the parameter
q$variant_label(c("r1i1p1f1", "r1i2p1f1"))

# negate the constraints
q$variant_label(!c("r1i1p1f1", "r1i2p1f1"))

# remove the parameter
q$variant_label(NULL)

## End(Not run)

## ------------------------------------------------
## Method `EsgfQuery$nominal_resolution`
## ------------------------------------------------

## Not run:
# get current value
q$nominal_resolution()

# set the parameter
q$nominal_resolution(c("100 km", "1x1 degree"))

# negate the constraints
q$nominal_resolution(!c("100 km", "1x1 degree"))

# remove the parameter
q$nominal_resolution(NULL)

## End(Not run)

## ------------------------------------------------
## Method `EsgfQuery$data_node`
## ------------------------------------------------

## Not run:
# get current value
q$data_node()

# set the parameter
q$data_node("esg.lasg.ac.cn")
```

```
# negate the constraints
q$data_node(!"esg.lasg.ac.cn")

# remove the parameter
q$data_node(NULL)

## End(Not run)

## ------------------------------------------------
## Method `EsgfQuery$facets`
## ------------------------------------------------

## Not run:
# get current value
q$facets()

# set the facets
q$facets(c("activity_id", "source_id"))

# use all available facets
q$facets("*")

## End(Not run)

## ------------------------------------------------
## Method `EsgfQuery$fields`
## ------------------------------------------------

## Not run:
# get current value
q$fields()

# set the fields
q$fields(c("activity_id", "source_id"))

# use all available fields
q$fields("*")

# remove the parameter
# act the same as above because the default `fields` in ESGF search
# services is `*` if `fields` is not specified
q$fields(NULL)

## End(Not run)

## ------------------------------------------------
## Method `EsgfQuery$shards`
## ------------------------------------------------

## Not run:
# get current value
q$shards()
```

```
# set the parameter
q$shards("localhost:8983/solr/datasets")

# negate the constraints
q$shards(!"localhost:8983/solr/datasets")

# only applicable for distributed queries
q$distrib(FALSE)$shards("localhost:8983/solr/datasets") # Error

# remove the parameter
q$shards(NULL)

## End(Not run)

## ------------------------------------------------
## Method `EsgfQuery$replica`
## ------------------------------------------------

## Not run:
# get current value
q$replica()

# set the parameter
q$replica(TRUE)

# remove the parameter
q$replica(NULL)

## End(Not run)

## ------------------------------------------------
## Method `EsgfQuery$latest`
## ------------------------------------------------

## Not run:
# get current value
q$latest()

# set the parameter
q$latest(TRUE)

## End(Not run)

## ------------------------------------------------
## Method `EsgfQuery$type`
## ------------------------------------------------

## Not run:
# get current value
q$type()

# set the parameter
q$type("Dataset")
```

```
## End(Not run)

## ------------------------------------------------
## Method `EsgfQuery$limit`
## ------------------------------------------------

## Not run:
# get current value
q$limit()

# set the parameter
q$limit(10L)

# `limit` is reset to 10,000 if input is greater than that
q$limit(10000L) # warning

## End(Not run)

## ------------------------------------------------
## Method `EsgfQuery$offset`
## ------------------------------------------------

## Not run:
# get current value
q$offset()

# set the parameter
q$offset(0L)

## End(Not run)

## ------------------------------------------------
## Method `EsgfQuery$distrib`
## ------------------------------------------------

## Not run:
# get current value
q$distrib()

# set the parameter
q$distrib(TRUE)

## End(Not run)

## ------------------------------------------------
## Method `EsgfQuery$params`
## ------------------------------------------------

## Not run:
# get current values
# default is an empty list (`list()`)
q$params()
```

```
# set the parameter
q$params(table_id = c("3hr", "day"), member_id = "00")
q$params()

# reset existing parameters
q$frequency("day")
q$params(frequency = "mon")
q$frequency() # frequency value has been changed using $params()

# negating the constraints is also supported
q$params(table_id = !c("3hr", "day"))

# use NULL to remove all parameters
q$params(NULL)$params()

## End(Not run)

## ------------------------------------------------
## Method `EsgfQuery$url`
## ------------------------------------------------

## Not run:
q$url()

# get the wget script URL
q$url(wget = TRUE)

# You can download the wget script using the URL directly. For
# example, the code below downloads the script and save it as
# 'wget.sh' in R's temporary folder:
download.file(q$url(TRUE), file.path(tempdir(), "wget.sh"), mode = "wb")


## End(Not run)

## ------------------------------------------------
## Method `EsgfQuery$count`
## ------------------------------------------------

## Not run:
# get the total number of matched records
q$count(NULL) # or q$count(facets = FALSE)

# count records for specific facets
q$facets(c("activity_id", "source_id"))$count()

# same as above
q$count(facets = c("activity_id", "source_id"))

## End(Not run)

## ------------------------------------------------
```

```
## Method `EsgfQuery$collect`
## ------------------------------------------------

## Not run:
q$fields("source_id")
q$collect()

## End(Not run)

## ------------------------------------------------
## Method `EsgfQuery$response`
## ------------------------------------------------

## Not run:
q$response()

## End(Not run)

## ------------------------------------------------
## Method `EsgfQuery$print`
## ------------------------------------------------

## Not run:
q$print()

## End(Not run)
```

---

esgf_query *Query CMIP6 data using ESGF search RESTful API*

Description

Query CMIP6 data using ESGF search RESTful API

Usage

```
esgf_query(
  activity = "ScenarioMIP",
 variable = c("tas", "tasmax", "tasmin", "hurs", "hursmax", "hursmin", "pr", "rsds",
    "rlds", "psl", "sfcWind", "clt"),
  frequency = "day",
  experiment = c("ssp126", "ssp245", "ssp370", "ssp585"),
 source = c("AWI-CM-1-1-MR", "BCC-CSM2-MR", "CESM2", "CESM2-WACCM", "EC-Earth3",
    "EC-Earth3-Veg", "GFDL-ESM4", "INM-CM4-8", "INM-CM5-0", "MPI-ESM1-2-HR",
    "MRI-ESM2-0"),
  variant = "r1i1p1f1",
  replica = FALSE,
  latest = TRUE,
  resolution = c("100 km", "50 km"),
```

```
  type = "Dataset",
  limit = 10000L,
  data_node = NULL
)
```

Arguments

activity	A character vector indicating activity identifiers. Default: `"ScenarioMIP"`. Possible values: • `"AerChemMIP"`: Aerosols and Chemistry Model Intercomparison Project, • `"C4MIP"`: Coupled Climate Carbon Cycle Model Intercomparison Project, • `"CDRMIP"`: Carbon Dioxide Removal Model Intercomparison Project, • `"CFMIP"`: Cloud Feedback Model Intercomparison Project, • `"CMIP"`: CMIP DECK: 1pctCO2, abrupt4xCO2, amip, esm-piControl, esm-historical, historical, and piControl experiments, • `"CORDEX"`: Coordinated Regional Climate Downscaling Experiment, • `"DAMIP"`: Detection and Attribution Model Intercomparison Project, • `"DCPP"`: Decadal Climate Prediction Project, • `"DynVarMIP"`: Dynamics and Variability Model Intercomparison Project, • `"FAFMIP"`: Flux-Anomaly-Forced Model Intercomparison Project, • `"GMMIP"`: Global Monsoons Model Intercomparison Project, • `"GeoMIP"`: Geoengineering Model Intercomparison Project, • `"HighResMIP"`: High-Resolution Model Intercomparison Project, • `"ISMIP6"`: Ice Sheet Model Intercomparison Project for CMIP6, • `"LS3MIP"`: Land Surface, Snow and Soil Moisture, • `"LUMIP"`: Land-Use Model Intercomparison Project, • `"OMIP"`: Ocean Model Intercomparison Project, • `"PAMIP"`: Polar Amplification Model Intercomparison Project, • `"PMIP"`: Palaeoclimate Modelling Intercomparison Project, • `"RFMIP"`: Radiative Forcing Model Intercomparison Project, • `"SIMIP"`: Sea Ice Model Intercomparison Project, • `"ScenarioMIP"`: Scenario Model Intercomparison Project, • `"VIACSAB"`: Vulnerability, Impacts, Adaptation and Climate Services Advisory Board, • `"VolMIP"`: Volcanic Forcings Model Intercomparison Project
variable	A character vector indicating variable identifiers. The 12 most related variables for EPW are set as defaults. If `NULL`, all possible variables are returned. Default: `c("tas", "tasmax", "tasmin", "hurs", "hursmax", "hursmin", "psl", "rss", "rls", "sfcWind", "pr", "clt")`, where: • `tas`: Near-surface (usually, 2 meter) air temperature, units: `K`. • `tasmax`: Maximum near-surface (usually, 2 meter) air temperature, units: `K`. • `tasmin`: Minimum near-surface (usually, 2 meter) air temperature, units: `K`. • `hurs`: Near-surface relative humidity, units: `%`.

- hursmax: Maximum near-surface relative humidity, units: %.
- hursmin: Minimum near-surface relative humidity, units: %.
- psl: Sea level pressure, units: Pa.
- rsds: Surface downwelling shortwave radiation, units: W m-2.
- rlds: Surface downwelling longwave radiation, units: W m-2.
- sfcWind: Near-surface (usually, 10 meters) wind speed, units: m s-1.
- pr: Precipitation, units: kg m-2 s-1.
- clt: Total cloud area fraction for the whole atmospheric column, as seen from the surface or the top of the atmosphere. Units: %.

frequency — A character vector of sampling frequency. If NULL, all possible frequencies are returned. Default: "day". Possible values:

- "1hr": sampled hourly,
- "1hrCM": monthly-mean diurnal cycle resolving each day into 1-hour means,
- "1hrPt": sampled hourly, at specified time point within an hour,
- "3hr": sampled every 3 hours,
- "3hrPt": sampled 3 hourly, at specified time point within the time period,
- "6hr": sampled every 6 hours,
- "6hrPt": sampled 6 hourly, at specified time point within the time period,
- "day": daily mean samples,
- "dec": decadal mean samples,
- "fx": fixed (time invariant) field,
- "mon": monthly mean samples,
- "monC": monthly climatology computed from monthly mean samples,
- "monPt": sampled monthly, at specified time point within the time period,
- "subhrPt": sampled sub-hourly, at specified time point within an hour,
- "yr": annual mean samples,
- "yrPt": sampled yearly, at specified time point within the time period

experiment — A character vector indicating root experiment identifiers. The **Tier-1** experiment of activity ScenarioMIP are set as defaults. If NULL, all possible experiment are returned. Default: c("ssp126", "ssp245", "ssp370", "ssp585").

source — A character vector indicating model identifiers. Defaults are set to 11 sources which give outputs of all 4 experiment of activity ScenarioMIP with daily frequency, i.e. "AWI-CM-1-1-MR", "BCC-CSM2-MR", "CESM2", "CESM2-WACCM", "EC-Earth3", "EC-Earth3-Veg", "GFDL-ESM4", "INM-CM4-8", "INM-CM5-0", "MPI-ESM1-2-HR" and "MRI-ESM2-0". If NULL, all possible sources are returned.

variant — A character vector indicating label constructed from 4 indices stored as global attributes in format r<k>i<l>p<m>f<n> described below. Default: "r1i1p1f1". If NULL, all possible variants are returned.

- r: realization_index (<k>) = realization number (integer >0)
- i: initialization_index (<l>) = index for variant of initialization method (integer >0)
- p: physics_index (<m>) = index for model physics variant (integer >0)

- f: forcing_index (<n>) = index for variant of forcing (integer >0)

replica	Whether the record is the "master" copy, or a replica. Use FALSE to return only originals and TRUE to return only replicas. Use NULL to return both the master and the replicas. Default: FALSE.
latest	Whether the record is the latest available version, or a previous version. Use TRUE to return only the latest version of all records and FALSE to return previous versions. Default: FALSE.
resolution	A character vector indicating approximate horizontal resolution. Default: c("50 km", "100 km"). If NULL, all possible resolutions are returned.
type	A single string indicating the intrinsic type of the record. Should be either "Dataset" or "File". Default: "Dataset".
limit	An integer indicating the maximum of matched records to return. Should be <= 10,000. Default: 10000.
data_node	A character vector indicating data nodes to be queried. Default to NULL, which means all possible data nodes.

Details

The Earth System Grid Federation (ESGF) is an international collaboration for the software that powers most global climate change research, notably assessments by the Intergovernmental Panel on Climate Change (IPCC).

The ESGF search service exposes a RESTful URL that can be used by clients to query the contents of the underlying search index, and return results matching the given constraints. With the distributed capabilities of the ESGF search, the URL at any Index Node can be used to query that Node only, or all Nodes in the ESGF system. esgf_query() uses the LLNL (Lawrence Livermore National Laboratory) Index Node.

The core Controlled Vocabularies (CVs) for use in CMIP6, including all activities, experiment, sources (GCMs), frequencies can be found at the WCRP-CMIP/CMIP6_CVs GitHub repo.

Value

A data.table::data.table with an attribute named response which is a list converted from json response. If no matched data is found, an empty data.table is returned. Otherwise, the columns of returned data varies based on the type:

- If "Dataset", returned columns are:

No.	Column	Type	Description
1	dataset_id	Character	Dataset universal identifier
2	mip_era	Character	Activity's associated CMIP cycle. Will always be "CMIP6"
3	activity_drs	Character	Activity DRS (Data Reference Syntax)
4	institution_id	Character	Institution identifier
5	source_id	Character	Model identifier
6	experiment_id	Character	Root experiment identifier
7	member_id	Character	A compound construction from sub_experiment_id and variant_label
8	table_id	Character	Table identifier, i.e. sampling frequency identifier
9	frequency	Character	Sampling frequency

10	grid_label	Character	Grid identifier
11	version	Character	Approximate date of model output file
12	nominal_resolution	Character	Approximate horizontal resolution
13	variable_id	Character	Variable identifier
14	variable_long_name	Character	Variable long name
15	variable_units	Character	Units of variable
16	data_node	Character	Data node to download the model output file
17	dataset_pid	Character	A unique string that helps identify the dataset

- If "File", returned columns are:

No.	Column	Type	Description
1	file_id	Character	Model output file universal identifier
2	dataset_id	Character	Dataset universal identifier
3	mip_era	Character	Activity's associated CMIP cycle. Will always be "CMIP6"
4	activity_drs	Character	Activity DRS (Data Reference Syntax)
5	institution_id	Character	Institution identifier
6	source_id	Character	Model identifier
7	experiment_id	Character	Root experiment identifier
8	member_id	Character	A compound construction from sub_experiment_id and variant_label
9	table_id	Character	Table identifier, i.e. sampling frequency identifier
10	frequency	Character	Sampling frequency
11	grid_label	Character	Grid identifier
12	version	Character	Approximate date of model output file
13	nominal_resolution	Character	Approximate horizontal resolution
14	variable_id	Character	Variable identifier
15	variable_long_name	Character	Variable long name
16	variable_units	Character	Units of variable
17	datetime_start	POSIXct	Start date and time of simulation
18	datetime_end	POSIXct	End date and time of simulation
19	file_size	Character	Model output file size in Bytes
20	data_node	Character	Data node to download the model output file
21	file_url	Character	Model output file download url from HTTP server
22	tracking_id	Character	A unique string that helps identify the output file

References

https://github.com/ESGF/esgf.github.io/wiki/ESGF_Search_REST_API

Examples

```
## Not run:
esgf_query(variable = "rss", experiment = "ssp126", resolution = "100 km", limit = 1)

esgf_query(variable = "rss", experiment = "ssp126", type = "File", limit = 1)

## End(Not run)
```

`extract_data` *Extract data*

Description

`extract_data()` takes an `epw_cmip6_coord` object generated using `match_coord()` and extracts CMIP6 data using the coordinates and years of interest specified.

Usage

```
extract_data(
  coord,
  years = NULL,
  unit = FALSE,
  out_dir = NULL,
  by = NULL,
  keep = is.null(out_dir),
  compress = 100
)
```

Arguments

`coord`	An `epw_cmip6_coord` object created using `match_coord()`
`years`	An integer vector indicating the target years to be included in the data file. All other years will be excluded. If `NULL`, no subsetting on years will be performed. Default: `NULL`.
`unit`	If `TRUE`, units will be added to values using `units::set_units()`.
`out_dir`	The directory to save extracted data using `fst::write_fst()`. If `NULL`, all data will be kept in memory by default. Default: `NULL`.
`by`	A character vector of variable names used to split data during extraction. Should be a subset of: • `"experiment"`: root experiment identifiers • `"source"`: model identifiers • `"variable"`: variable identifiers • `"activity"`: activity identifiers • `"frequency"`: sampling frequency • `"variant"`: variant label • `"resolution"`: approximate horizontal resolution If `NULL` and `out_dir` is given, file name `data.fst` will be used. Default: `NULL`.
`keep`	Whether keep extracted data in memory. Default: `TRUE` if `out_dir` is `NULL`, and `FALSE` otherwise.
`compress`	A single integer in the range 0 to 100, indicating the amount of compression to use. Lower values mean larger file sizes. Default: `100`.

Details

`extract_data()` supports common calendars, including `365_day` and `360_day`, thanks to the PCICt package.

`extract_data()` uses future.apply underneath. You can use your preferable future backend to speed up data extraction in parallel. By default, `extract_data()` uses `future::sequential` backend, which runs things in sequential.

Value

An `epw_cmip6_data` object, which is basically a list of 3 elements:

- `epw`: An eplusr::Epw object whose longitude and latitude are used to extract CMIP6 data. It is the same object as created in `match_coord()`
- `meta`: A list containing basic metadata of input EPW, including `city`, `state_province`, `country`, `latitude` and `longitude`.
- `data`: An empty `data.table::data.table()` if `keep` is `FALSE` or a `data.table::data.table()` of 14 columns if `keep` is `TRUE`:

No.	Column	Type	Description
1	`activity_drs`	Character	Activity DRS (Data Reference Syntax)
2	`institution_id`	Character	Institution identifier
3	`source_id`	Character	Model identifier
4	`experiment_id`	Character	Root experiment identifier
5	`member_id`	Character	A compound construction from `sub_experiment_id` and `variant_label`
6	`table_id`	Character	Table identifier
7	`lon`	Double	Longitude of extracted location
8	`lat`	Double	Latitude of extracted location
9	`dist`	Double	The spherical distance in km between EPW location and grid coordinates
10	`datetime`	POSIXct	Datetime for the predicted value
11	`variable`	Character	Variable identifier
12	`description`	Character	Variable long name
13	`units`	Character	Units of variable
14	`value`	Double	The actual predicted value

Examples

```
## Not run:
coord <- match_coord("path_to_an_EPW")
extract_data(coord, years = 2030:2060)

## End(Not run)
```

`future_epw` *Create future EPW files using morphed data*

Description

Create future EPW files using morphed data

Usage

```
future_epw(
  morphed,
  by = c("experiment", "source", "interval"),
  dir = ".",
  separate = TRUE,
  overwrite = FALSE,
  full = FALSE
)
```

Arguments

morphed	An epw_cmip6_morphed object created using morphing_epw().
by	A character vector of columns to be used as grouping variables when creating EPW files. Should be a subset of: • "experiment": root experiment identifiers • "source": model identifiers • "variable": variable identifiers • "activity": activity identifiers • "frequency": sampling frequency • "variant": variant label • "resolution": approximate horizontal resolution • "longitude": averaged longitude of input data • "latitude": averaged latitude of input data
dir	The parent directory to save the generated EPW files. If not exist, it will be created first. Default: ".", i.e., current working directory.
separate	If TRUE, each EPW file will be saved into a separate folder using grouping variables specified in by.
overwrite	If TRUE, overwrite existing files if they exist. Default: FALSE.
full	If TRUE, a data.table containing information about how the data are split and also the generated future EPWs and their paths are returned. Default: FALSE.

Value

If full is FALSE, which is the default, a list of generated eplusr::Epw objects, invisibly. Otherwise, a data.table with columns:

- specified by the by value
- epw: a list of eplusr::Epw
- path: full paths of the generated EPW files

get_data_dir *Get the path of directory where epwshiftr data is stored*

Description

If option `epwshiftr.dir` is set, use it. Otherwise, get package data storage directory using `rappdirs::user_data_dir()`.

Usage

```
get_data_dir()
```

Value

A single string.

Examples

```
options(epwshiftr.dir = tempdir())
get_data_dir()
```

get_data_node *Get data nodes which store CMIP6 output*

Description

Get data nodes which store CMIP6 output

Usage

```
get_data_node(speed_test = FALSE, timeout = 3)
```

Arguments

`speed_test`	If `TRUE`, use `pingr::ping()` to perform connection speed test on each data node. A `ping` column is appended in returned data.table which stores each data node response in milliseconds. This feature needs pingr package already installed. Default: `FALSE`.
`timeout`	Timeout for a ping response in seconds. Default: `3`.

Value

A `data.table::data.table()` of 2 or 3 (when `speed_test` is `TRUE`) columns:

Column	Type	Description
`data_node`	character	Web address of data node
`status`	character	Status of data node. `"UP"` means OK and `"DOWN"` means currently not available
`ping`	double	Data node response in milliseconds during speed test

Examples

```
## Not run:
get_data_node()

## End(Not run)
```

`init_cmip6_index` *Build CMIP6 experiment output file index*

Description

`init_cmip6_index()` will search the CMIP6 model output file using `esgf_query()` , return a `data.table::data.table()` containing the actual NetCDF file url to download, and store it into user data directory for future use.

Usage

```
init_cmip6_index(
  activity = "ScenarioMIP",
 variable = c("tas", "tasmax", "tasmin", "hurs", "hursmax", "hursmin", "pr", "rsds",
    "rlds", "psl", "sfcWind", "clt"),
  frequency = "day",
  experiment = c("ssp126", "ssp245", "ssp370", "ssp585"),
 source = c("AWI-CM-1-1-MR", "BCC-CSM2-MR", "CESM2", "CESM2-WACCM", "EC-Earth3",
    "EC-Earth3-Veg", "GFDL-ESM4", "INM-CM4-8", "INM-CM5-0", "MPI-ESM1-2-HR",
    "MRI-ESM2-0"),
  variant = "r1i1p1f1",
  replica = FALSE,
  latest = TRUE,
  resolution = c("100 km", "50 km"),
  limit = 10000L,
  data_node = NULL,
  years = NULL,
  save = FALSE
)
```

Arguments

`activity` A character vector indicating activity identifiers. Default: `"ScenarioMIP"`. Possible values:

- `"AerChemMIP"`: Aerosols and Chemistry Model Intercomparison Project,
- `"C4MIP"`: Coupled Climate Carbon Cycle Model Intercomparison Project,
- `"CDRMIP"`: Carbon Dioxide Removal Model Intercomparison Project,
- `"CFMIP"`: Cloud Feedback Model Intercomparison Project,

- "CMIP": CMIP DECK: 1pctCO2, abrupt4xCO2, amip, esm-piControl, esm-historical, historical, and piControl experiments,
- "CORDEX": Coordinated Regional Climate Downscaling Experiment,
- "DAMIP": Detection and Attribution Model Intercomparison Project,
- "DCPP": Decadal Climate Prediction Project,
- "DynVarMIP": Dynamics and Variability Model Intercomparison Project,
- "FAFMIP": Flux-Anomaly-Forced Model Intercomparison Project,
- "GMMIP": Global Monsoons Model Intercomparison Project,
- "GeoMIP": Geoengineering Model Intercomparison Project,
- "HighResMIP": High-Resolution Model Intercomparison Project,
- "ISMIP6": Ice Sheet Model Intercomparison Project for CMIP6,
- "LS3MIP": Land Surface, Snow and Soil Moisture,
- "LUMIP": Land-Use Model Intercomparison Project,
- "OMIP": Ocean Model Intercomparison Project,
- "PAMIP": Polar Amplification Model Intercomparison Project,
- "PMIP": Palaeoclimate Modelling Intercomparison Project,
- "RFMIP": Radiative Forcing Model Intercomparison Project,
- "SIMIP": Sea Ice Model Intercomparison Project,
- "ScenarioMIP": Scenario Model Intercomparison Project,
- "VIACSAB": Vulnerability, Impacts, Adaptation and Climate Services Advisory Board,
- "VolMIP": Volcanic Forcings Model Intercomparison Project

`variable` A character vector indicating variable identifiers. The 12 most related variables for EPW are set as defaults. If NULL, all possible variables are returned. Default: c("tas", "tasmax", "tasmin", "hurs", "hursmax", "hursmin", "psl", "rss", "rls", "sfcWind", "pr", "clt"), where:

- tas: Near-surface (usually, 2 meter) air temperature, units: K.
- tasmax: Maximum near-surface (usually, 2 meter) air temperature, units: K.
- tasmin: Minimum near-surface (usually, 2 meter) air temperature, units: K.
- hurs: Near-surface relative humidity, units: %.
- hursmax: Maximum near-surface relative humidity, units: %.
- hursmin: Minimum near-surface relative humidity, units: %.
- psl: Sea level pressure, units: Pa.
- rsds: Surface downwelling shortwave radiation, units: W m-2.
- rlds: Surface downwelling longwave radiation, units: W m-2.
- sfcWind: Near-surface (usually, 10 meters) wind speed, units: m s-1.
- pr: Precipitation, units: kg m-2 s-1.
- clt: Total cloud area fraction for the whole atmospheric column, as seen from the surface or the top of the atmosphere. Units: %.

`frequency` A character vector of sampling frequency. If NULL, all possible frequencies are returned. Default: "day". Possible values:

- "1hr": sampled hourly,

- "1hrCM": monthly-mean diurnal cycle resolving each day into 1-hour means,
- "1hrPt": sampled hourly, at specified time point within an hour,
- "3hr": sampled every 3 hours,
- "3hrPt": sampled 3 hourly, at specified time point within the time period,
- "6hr": sampled every 6 hours,
- "6hrPt": sampled 6 hourly, at specified time point within the time period,
- "day": daily mean samples,
- "dec": decadal mean samples,
- "fx": fixed (time invariant) field,
- "mon": monthly mean samples,
- "monC": monthly climatology computed from monthly mean samples,
- "monPt": sampled monthly, at specified time point within the time period,
- "subhrPt": sampled sub-hourly, at specified time point within an hour,
- "yr": annual mean samples,
- "yrPt": sampled yearly, at specified time point within the time period

`experiment` A character vector indicating root experiment identifiers. The **Tier-1** experiment of activity ScenarioMIP are set as defaults. If NULL, all possible experiment are returned. Default: c("ssp126", "ssp245", "ssp370", "ssp585").

`source` A character vector indicating model identifiers. Defaults are set to 11 sources which give outputs of all 4 experiment of activity ScenarioMIP with daily frequency, i.e. "AWI-CM-1-1-MR", "BCC-CSM2-MR", "CESM2", "CESM2-WACCM", "EC-Earth3", "EC-Earth3-Veg", "GFDL-ESM4", "INM-CM4-8", "INM-CM5-0", "MPI-ESM1-2-HR" and "MRI-ESM2-0". If NULL, all possible sources are returned.

`variant` A character vector indicating label constructed from 4 indices stored as global attributes in format r<k>i<l>p<m>f<n> described below. Default: "r1i1p1f1". If NULL, all possible variants are returned.

- r: realization_index (<k>) = realization number (integer >0)
- i: initialization_index (<l>) = index for variant of initialization method (integer >0)
- p: physics_index (<m>) = index for model physics variant (integer >0)
- f: forcing_index (<n>) = index for variant of forcing (integer >0)

`replica` Whether the record is the "master" copy, or a replica. Use FALSE to return only originals and TRUE to return only replicas. Use NULL to return both the master and the replicas. Default: FALSE.

`latest` Whether the record is the latest available version, or a previous version. Use TRUE to return only the latest version of all records and FALSE to return previous versions. Default: FALSE.

`resolution` A character vector indicating approximate horizontal resolution. Default: c("50 km", "100 km"). If NULL, all possible resolutions are returned.

`limit` An integer indicating the maximum of matched records to return. Should be <= 10,000. Default: 10000.

`data_node` A character vector indicating data nodes to be queried. Default to NULL, which means all possible data nodes.

years	An integer vector indicating the target years to be include in the data file. All other years will be excluded. If NULL, no subsetting on years will be performed. Default: NULL.
save	If TRUE, the results will be saved into user data directory. Default: FALSE.

Details

For details on where the file index is stored, see rappdirs::user_data_dir().

Value

A data.table::data.table with 22 columns:

No.	Column	Type	Description
1	file_id	Character	Model output file universal identifier
2	dataset_id	Character	Dataset universal identifier
3	mip_era	Character	Activity's associated CMIP cycle. Will always be "CMIP6"
4	activity_drs	Character	Activity DRS (Data Reference Syntax)
5	institution_id	Character	Institution identifier
6	source_id	Character	Model identifier
7	experiment_id	Character	Root experiment identifier
8	member_id	Character	A compound construction from sub_experiment_id and variant_label
9	table_id	Character	Table identifier
10	frequency	Character	Sampling frequency
11	grid_label	Character	Grid identifier
12	version	Character	Approximate date of model output file
13	nominal_resolution	Character	Approximate horizontal resolution
14	variable_id	Character	Variable identifier
15	variable_long_name	Character	Variable long name
16	variable_units	Character	Units of variable
17	datetime_start	POSIXct	Start date and time of simulation
18	datetime_end	POSIXct	End date and time of simulation
19	file_size	Character	Model output file size in Bytes
20	data_node	Character	Data node to download the model output file
21	dataset_pid	Character	A unique string that helps identify the dataset
22	tracking_id	Character	A unique string that helps identify the output file

Note

Argument limit will only apply to Dataset query. init_cmip6_index() will try to get all model output files which match the dataset id.

Examples

```
## Not run:
init_cmip6_index()

## End(Not run)
```

load_cmip6_index *Load previously stored CMIP6 experiment output file index*

Description

Load previously stored CMIP6 experiment output file index

Usage

```
load_cmip6_index(force = FALSE)
```

Arguments

force	If TRUE, read the index file. Otherwise, return the cached index if exists. Default: FALSE.

Value

A data.table::data.table with 20 columns. For detail description on column, see `init_cmip6_index()`.

Examples

```
## Not run:
load_cmip6_index()

## End(Not run)
```

match_coord *Match coordinates of input EPW in the CMIP6 output file database*

Description

`match_coord()` takes an EPW and uses its longitude and latitude to calculate the distance between the EPW location and the global grid points in NetCDF files.

Usage

```
match_coord(epw, threshold = list(lon = 1, lat = 1), max_num = NULL)
```

Arguments

epw	Possible values: • A file path of EPW file • An eplusr::Epw object • A regular expression used to search locations in EnergyPlus Weather Database, e.g. "los angeles.*tmy3". You will be asked to select a matched EPW to download and read. It will be saved into `tempdir()`. Note that the search is case-insensitive
threshold	A list of 2 elements `lon` and `lat` specifying the absolute distance threshold used for subsetting longitude and latitude value for calculating distances. If `NULL`, no subsetting is performed and the distances between the target location and all grid points are calculated. This is useful set the `threshold` value to exclude some points that are absolute too far away from the target location. Default: `list(lon = 1.0, lat = 1.0)`
max_num	The maximum number of grid points to be matched. Default is `NULL`, which means no number limit and the total matched grid points are determined by the `threshold` input.

Details

`match_coord()` uses future.apply underneath. You can use your preferable future backend to speed up data extraction in parallel. By default, `match_coord()` uses `future::sequential` backend, which runs things in sequential.

Value

An `epw_cmip6_coord` object, which is basically a list of 3 elements:

- `epw`: An eplusr::Epw object parsed from input `epw` argument
- `meta`: A list containing basic meta data of input EPW, including `city`, `state_province`, `country`, `latitute` and `longitude`.
- `coord`: A `data.table::data.table()` which is basically CMIP6 index database with an appending new list column `coord` that contains matched latitudes and longitudes in each NetCDF file. Each element in `coord` is a `data.table::data.table()` of 6 columns describing the matched coordinates.
  - `index`: the indices of matched coordinates
  - `ind_lon`, `ind_lat`: The value indices of longitude or latitude in the NetCDF coordinate grids. These values are used to extract the corresponding variable values
  - `lon`, `lat`: the actual longitude or latitude in the NetCDF coordinate grids
  - `dist`: the distance in km between the coordinate values in NetCDF and input EPW

Geographical distance calculation

`match_coord()` calculates the geographical distances based formulas of spherical trigonometry:

$$\Delta X = \cos(\phi_2)\cos(\lambda_2) - \cos(\phi_1)\cos(\lambda_1)$$

$$\Delta Y = \cos(\phi_2)\sin(\lambda_2) - \cos(\phi_1)\sin(\lambda_1)$$

$$\Delta Z = \sin(\phi_2) - \sin(\phi_1)$$

$$C_h = \sqrt{(\Delta X)^2 + (\Delta Y)^2 + (\Delta Z)^2}$$

where phi is the latitude and $lambda$ is the longitude. This formula treats the Earth as a sphere. The geographical distance between points on the surface of a spherical Earth is $D = RC_h$.

For more details, please see this Wikipedia

Examples

```
## Not run:
# download an EPW from EnergyPlus website
epw <- eplusr::download_weather("los angeles.*TMY3", dir = tempdir(),
    type = "EPW", ask = FALSE)

match_coord(epw, threshold = list(lon = 1.0, lat = 1.0))

## End(Not run)
```

`morphing_epw` *Morphing EPW weather variables*

Description

`morphing_epw()` takes an `epw_cmip6_data` object generated using `extract_data()` and calculates future core EPW weather variables using Morphing Method.

Usage

```
morphing_epw(
  data,
  years = NULL,
  labels = NULL,
  methods = NULL,
  warning = FALSE
)
```

Arguments

data	An epw_cmip6_dataobject generated using extract_data()
years	An integer vector indicating the target years to be considered. If NULL, all years in input data will be considered. Default: NULL.
labels	A character or factor vector used for grouping input years. Usually are the outputs of base::cut(). labels should have the same length as years. If given, climate data of years grouped by labels will be averaged. Default: NULL.
methods	A named character giving the methods of morphing procedures of each variables. Possible variable names are tdb, rh, p, hor_ir, glob_rad, wind. Possible values are: "stretch", "shift" and "combined". For example: c(tdb = "stretch", rh = "shift"). "combined" is only applicable to tdb. The default morphing method for each variable is listed in the *Return* section. If NULL, the default methods will be used. Default: NULL.
warning	If TRUE, warnings will be issued for cases with input data less than a decade (10 years). This is because using data that only covers a short period of time may not be able to capture the average of future climate. Default: FALSE.

Details

The EPW weather variables that get morphed are listed in details.

Value

An epw_cmip6_morphed object, which is basically a list of 12 elements:

No.	Element	Type	Morphing Method	Description
1	epw	eplusr::Epw	N/A	The original EPW file used for morphing
2	tdb	data.table::data.table()	Stretch	Data of dry-bulb temperature after morphing
3	tdew	data.table::data.table()	Derived	Data of dew-point temperature after morphing
4	rh	data.table::data.table()	Stretch	Data of relative humidity after morphing
5	p	data.table::data.table()	Stretch	Data of atmospheric pressure after morphing
6	hor_ir	data.table::data.table()	Stretch	Data of horizontal infrared radiation from the sk
7	glob_rad	data.table::data.table()	Stretch	Data of global horizontal radiation after morphi
8	norm_rad	data.table::data.table()	Derived	Data of direct normal radiation after morphing
9	diff_rad	data.table::data.table()	Stretch	Data of diffuse horizontal radiation after morphi
10	wind	data.table::data.table()	Stretch	Data of wind speed after morphing
11	total_cover	data.table::data.table()	Derived	Data of total sky cover after morphing
12	opaque_cover	data.table::data.table()	Derived	Data of opaque sky cover after morphing

Each data.table::data.table() listed above contains 19 columns below or an empty data.table::data.table() if the corresponding variables cannot be found in the input epw_cmip6_data object.

No.	Column	Type	Description
1	activity_drs	Character	Activity DRS (Data Reference Syntax)
2	institution_id	Character	Institution identifier

3	source_id	Character	Model identifier
4	experiment_id	Character	Root experiment identifier
5	member_id	Character	A compound construction from sub_experiment_id and variant_label
6	table_id	Character	Table identifier
7	lon	Double	The **averaged** values of input longitude
8	lat	Double	The **averaged** values of input latitude
9	dist	Double	The **averaged** spherical distances in km between EPW location and grid coordinates
10	interval	Factor	The label value used to average raw input data
11	datetime	POSIXct	The datetime value with **fake year** generated by calling the Epw$data() method with t
12	year	Integer	The **original** year of the raw EPW data
13	month	Integer	The month value of the morphed data
14	day	Integer	The day of the morphed data
15	hour	Integer	The hour of the morphed data
16	minute	Integer	The minute of the morphed data
17	**Variable Name**	Double	The morphed data, where Variable Name is the corresponding EPW weather variable
18	delta	Double	The shift factor. Will be NA for derived values
19	alpha	Double	The stretch factor. Will be NA for derived values

The Morphing procedure

Here *Morphing* is an algorithm proposed by Belcher etc. 2005 used to morph the present-day observed weather files (here the EPWs) to produce future climate weather files. The EPW data is used as the *'baseline climate'*.

The first step before morphing is to calculate the monthly means of climatological variables in the EPW file, denoted by $< x_0 >_m$. The subscript '0' is to denote the present day weather record, and 'm' is to denote the month.

The morphing involves three generic operations, i.e. 1) a shift; 2) a linear stretch (scaling factor); and 3) a shift and a stretch:

$$\text{Shift: } x = x_0 + \Delta x_m$$

$$\text{Stretch: } x = \alpha_m x_m$$

$$\text{Shift + Stretch: } x = x_0 + \Delta x_m + \alpha_m(x_0 - < x_0 >_m)$$

Shift:

If using a shift, for each month, a shift Δx_m is applied to x_0. Δx_m is the absolute change in the monthly mean value of the variable for the month m, i.e. $\Delta x_m =< x_0 >_m - < x >_m$. Here the monthly variance of the variable is unchanged.

Stretch:

If using a stretch, for each month, a stretch α_m is applied to x_0, where α_m is the fractional change in the monthly-mean value of a variable, i.e. $\alpha_m =< x >_m / < x_0 >_m$. In this case, the variance will be multiplied by to α_m^2

Combined Shift and Stretch:

When using a combined shift and stretch factor, both the mean and the variance will be switched off altogether.

For more details about morphing, please see (Belcher etc. 2005)

References

Belcher, S., Hacker, J., Powell, D., 2005. Constructing design weather data for future climates. Building Services Engineering Research and Technology 26, 49–61. https://doi.org/10.1191/0143624405bt112oa

---

`set_cmip6_index` *Set CMIP6 index*

Description

`set_cmip6_index()` takes a `data.table::data.table()` as input and set it as current index.

Usage

```
set_cmip6_index(index, save = FALSE)
```

Arguments

`index`	A `data.table::data.table()` containing the same column names and types as the output of `init_cmip6_index()`.
`save`	If `TRUE`, besides loaded index, the index file saved to data directory will be also updated. Default: `FALSE`.

Details

`set_cmip6_index()` is useful when `init_cmip6_index()` may give you too much cases of which only some are of interest.

Value

A `data.table::data.table()`.

summary_database *Summary CMIP6 model output file status*

Description

summary_database() scans the directory specified and returns a data.table() containing summary information about all the CMIP6 files available against the output file index loaded using load_cmip6_index().

Usage

```
summary_database(
  dir,
  by = c("activity", "experiment", "variant", "frequency", "variable", "source",
    "resolution"),
  mult = c("skip", "latest"),
  append = FALSE,
  miss = c("keep", "overwrite"),
  recursive = FALSE,
  update = FALSE,
  warning = TRUE
)
```

Arguments

dir	A single string indicating the directory where CMIP6 model output NetCDF files are stored.
by	The grouping column to summary the database status. Should be a subset of: • "experiment": root experiment identifiers • "source": model identifiers • "variable": variable identifiers • "activity": activity identifiers • "frequency": sampling frequency • "variant": variant label • "resolution": approximate horizontal resolution
mult	Actions when multiple files match a same case in the CMIP6 index. If "latest", the file with latest modification time will be used. If "skip", all matched files will be skip and this case will be kept as unmatched. Default: "skip".
append	If TRUE, status of CMIP6 files will only be updated if they are not found in previous summary. This is useful if CMIP6 files are stored in different directories. Default: FALSE.
miss	Actions when matched files in the previous summary do not exist when running current summary. Only applicable when append is set to TRUE. If "keep", the metadata for the missing output files will be kept. If "overwrite", existing metadata of those output will be first removed from the output file index and overwritten based on the newly matched files if possible. Default: "keep".

recursive	If TRUE, scan recursively into directories. Default: FALSE.
update	If TRUE, the output file index will be updated based on the matched NetCDF files in specified directory. If FALSE, only current loaded index will be updated, but the actual index database file saved in get_data_dir() will remain unchanged. Default: FALSE.
warning	If TRUE, warning messages will show when target files are missing or multiple files match a same case. Default: TRUE.

Details

The database here can be any directory that stores the NetCDF files for CMIP6 GCMs. It can be also be the same as get_data_dir() where epwshiftr stores the output file index, if you want to save the output file index and output files in the same place.

summary_database() uses the tracking_id, datetime_start and datetime_end global attributes of each NetCDF file to match against the output file index. So the names of NetCDF files do not necessarily follow the CMIP6 file name encoding.

summary_database() will append 5 columns in the CMIP6 output file index:

- file_path: the full path of matched NetCDF file for every case.

summary_database() uses future.apply underneath to speed up the data processing if applicable. You can use your preferable future backend to speed up data extraction in parallel. By default, summary_database() uses future::sequential backend, which runs things in sequential.

Value

A data.table::data.table() containing corresponding grouping columns plus:

Column	Type	Description
datetime_start	POSIXct	Start date and time of simulation
datetime_end	POSIXct	End date and time of simulation
file_num	Integer	Total number of file per group
file_size	Units (Mbytes)	Approximate total size of file
dl_num	Integer	Total number of file downloaded
dl_percent	Units (%)	Total percentage of file downloaded
dl_size	Units (Mbytes)	Total size of file downloaded

Also 2 extra data.table::data.table() are attached as **attributes**:

- not_found: A data.table::data.table() that contains metadata for those CMIP6 outputs that are listed in current CMIP6 output file index but the existing file paths are not valid now and cannot be found in current database.
- not_matched: A data.table::data.table() that contains metadata for those CMIP6 output files that are found in current database but not listed in current CMIP6 output file index.

For the meaning of grouping columns, see init_cmip6_index().

Examples

```
## Not run:
summary_database()

summary_database(by = "experiment")

## End(Not run)
```

Index