Package ‘STATcubeR’

January 20, 2025

Type Package

Title R Interface for the 'STATcube' REST API and Open Government Data

Version 1.0.0

Date 2024-11-29

Description Import data from the 'STATcube' REST API or from the open data portal of Statistics Austria. This package includes a client for API requests as well as parsing utilities for data which originates from 'STATcube'. Documentation about 'STATcubeR' is provided by several vignettes included in the package as well as on the public 'pkgdown' page at <https://statistikat.github.io/STATcubeR/>.

License GPL (>= 2)

URL https://statistikat.github.io/STATcubeR/, https://github.com/statistikat/STATcubeR

BugReports https://github.com/statistikat/STATcubeR/issues

Depends R (>= 3.5.0)

Imports cli (>= 3.4.1), httr, jsonlite, pillar (>= 1.5.0), vctrs (>= 0.5.2)

Suggests magrittr, spelling, data.tree, rappdirs, xml2, reactable, markdown

Encoding UTF-8

RoxygenNote 7.3.2

Language en-US

NeedsCompilation no

Author Bernhard Meindl [ctb, cre],
Alexander Kowarik [ctb] (<https://orcid.org/0000-0001-8598-4130>),
Gregor de Cillia [aut]

Maintainer Bernhard Meindl <Bernhard.Meindl@statistik.gv.at>

Repository CRAN

Date/Publication 2024-11-29 09:50:02 UTC

Contents

`od_cache` *Cache management for Open Data*

Description

Functions to inspect the contents of the current cache.

Usage

```
od_cache_summary(server = "ext")

od_downloads(server = "ext")
```

Arguments

`server`	the OGD-Server to use. `"ext"` for the external server (the default) or `"red"` for the editing server

Value

- `od_cache_summary()` provides an overview of all contents of the cache through a data.frame. It has one row for each dataset and returns a `data.frame` with# the following columns in which all file sizes are given in bytes.
  - `id` the dataset id
  - `updated` the last modified time for `${id}.json`

  - `json` the file size of `${id}.json`
  - `data` the file size of `${id}.csv`
  - `header` the file size of `${id}_HEADER.csv`
  - `fields` the total file size of all files belonging to fields (`{id}_C*.csv`).
  - `n_fields` the number of field files
- `od_downloads()` shows a download history for the current cache and returns a `data.frame` with the following columns:
  - `time` a timestamp for the download
  - `file` the filename
  - `downloaded` the download time in milliseconds

Examples

```
## make sure the cache is not empty
od_table("OGD_krebs_ext_KREBS_1")
od_table("OGD_veste309_Veste309_1")

## inspect
od_cache_summary()
od_downloads()
```

`od_catalogue` *Get a catalogue for OGD datasets*

Description

EXPERIMENTAL This function parses several json metadata files at once and combines them into a `data.frame` so the datasets can easily be filtered based on categorizations, tags, number of classifications, etc.

Usage

```
od_catalogue(server = "ext", local = TRUE)
```

Arguments

`server`	the OGD-server to be used. `"ext"` (the default) for the external server or `prod` for the production server
`local`	If `TRUE` (the default), the catalogue is created based on cached json metadata. Otherwise, the cache is updated prior to creating the catalogue using a "bulk-download" for metadata files.

Details

The naming, ordering and choice of the columns is likely to change.

Value

a `data.frame` with the following structure

Column	**Type**	**Description**
title	`chr`	Title of the dataset
measures	`int`	Number of measure variables
fields	`int`	Number of classification fields
modified	`datetime`	Timestamp when the dataset was last modified
created	`datetime`	Timestamp when the dataset was created
database	`chr`	ID of the corresponding STATcube database
title_en	`chr`	English title
notes	`chr`	Description for the dataset
frequency	`chr`	How often is the dataset updated?
category	`chr`	Category of the dataset
tags	`list<chr>`	tags assigned to the dataset
json	`list<od_json>`	Full json metadata

The type `datetime` refers to the `POSIXct` format as returned by `Sys.time()`. The last column `"json"` contains the full json metadata as returned by `od_json()`.

Examples

```
catalogue <- od_catalogue()
catalogue
table(catalogue$update_frequency)
table(catalogue$categorization)
catalogue[catalogue$categorization == "Gesundheit", 1:4]
catalogue[catalogue$measures >= 70, 1:3]
catalogue$json[[1]]
head(catalogue$database)
```

`od_list` *List available Opendata datasets*

Description

`od_list()` returns a `data.frame` containing all datasets published at data.statistik.gv.at

Usage

```
od_list(unique = TRUE, server = c("ext", "red"))
```

Arguments

`unique`	some datasets are published under multiple groups. They will only be listed once with the first group they appear in unless this parameter is set to `FALSE`.
`server`	the open data server to use. Either `ext` for the external server (the default) or `red` for the editing server. The editing server is only accessible for employees of Statistics Austria

Value

a `data.frame` with two columns

- `"category"`: Grouping under which a dataset is listed
- `"id"`: Name of the dataset which can later be used in `od_table()`
- `"label"`: Description of the dataset

Examples

```
df <- od_list()
df
subset(df, category == "Bildung und Forschung")
# use an id to load a dataset
od_table("OGD_fhsstud_ext_FHS_S_1")
```

`od_resource` *Resource management for open.data*

Description

Helper functions for caching and parsing open.data resources.

Usage

```
od_cache_dir(dir = NULL)

od_cache_clear(id, server = "ext")

od_cache_file(id, suffix = NULL, timestamp = NULL, ..., server = "ext")

od_resource(id, suffix = NULL, timestamp = NULL, server = "ext")

od_json(id, timestamp = Sys.time() - 3600, server = "ext")

od_resource_all(id, json = od_json(id), server = "ext")
```

Arguments

dir	If NULL, the cache directory is returned. Otherwise, the cache directory will be updated to dir.
id	A database id
server	the OGD-Server to use to load update the resources in case they are outdated. "ext" for the external server (the default) od "red" for the editing server.
suffix	A suffix for the resource: "HEADER" or a field code.
timestamp	A timestamp in POSIXct format. If provided, the cached resource will be updated if it is older than that value. Otherwise it will be downloaded only if it does not exist in the cache.
...	For internal use
json	The JSON file belonging to the dataset

Details

`od_cache_clear(id)` removes all files belonging to the specified id.

By default, downloaded json files will "expire" in one hour or 3600 seconds. That is, if a json is requested, it will be reused from the cache unless the `file.mtime()` is more than one hour behind `Sys.time()`.

Value

For `od_cache_file()` and `od_resource()`, the returned objects contain a hidden attribute `attr(., "od")` about the time used for downloading and parsing the resource. `od_resource_all()` converts these hidden attribute into columns.

Examples

```
# get the current cache directory
od_cache_dir()

# Get paths to cached files
od_cache_file("OGD_veste309_Veste309_1")
od_cache_file("OGD_veste309_Veste309_1", "C-A11-0")

# get a parsed verison of the resource
od_resource("OGD_veste309_Veste309_1", "C-A11-0")

# get json metadata about a dataset
od_json('OGD_veste309_Veste309_1')

# Bundle all resources
od_resource_all("OGD_veste309_Veste309_1")
```

od_revisions *Get OGD revisions*

Description

Use the `/revision` endpoint of the OGD server to get a list of all datasets that have changed since a certain timestamp.

Usage

```
od_revisions(since = NULL, exclude_ext = TRUE, server = "ext")
```

Arguments

`since`	(optional) A timestamp. If supplied, only datasets updated later will be returned. Otherwise, all datasets are returned. Can be in either one of the following formats • a native R time type that is compatible with `strftime()` such as the return values of `Sys.Date()`, `Sys.time()` and `file.mtime()`. • a string of the form `YYYY-MM-DD` to specify a day. • a string of the form `YYYY-MM-DDThh:mm:ss` to specify a day and a time.
`exclude_ext`	If `TRUE` (default) exclude all results that have `OGDEXT_` as a prefix
`server`	the open data server to use. Either `ext` for the external server (the default) or `red` for the editing server. The editing server is only accessible for employees of Statistics Austria

Value

a character vector with dataset ids

Examples

```
# get all datasets (including OGDEXT_*)
ids <- od_revisions(exclude_ext = FALSE)
ids
sample(ids, 6)

# get all the datasets since the fifteenth of august
od_revisions("2022-09-15")
```

od_table *Create a table-instance from an open-data dataset*

Description

`od_table(id)` returns an `R6`-class object containing all relevant data and metadata from https://data.statistik.gv.at/data/

Usage

```
od_table(id, language = NULL, server = "ext")
```

Arguments

`id`	the id of the dataset that should be accessed
`language`	language to be used for labeling. `"en"` or `"de"`
`server`	the OGD-server to be used. `"ext"` (the default) for the external server or `prod` for the production server

Value

The returned objects is of class `sc_table` and inherits several parsing methods from sc_data. See od_table_class for the full class documentation.

Components

Component	**Corresponding File on Server**
`$data`	`https://data.statistik.gv.at/data/${id}.csv`
`$header`	`https://data.statistik.gv.at/data/${id}_HEADER.csv`
`$field(code)`	`https://data.statistik.gv.at/data/${id}_${code}.csv`
`$json`	`https://data.statistik.gv.at/ogd/json?dataset=${id}`

Examples

```
x <- od_table("OGD_krebs_ext_KREBS_1")

## metadata
x
x$meta
x$field("Sex")
x$field(3)

## data
x$data
x$tabulate()
```

```
## tabulation: see `?sc_tabulate` for more examples
x$tabulate("Reporting year", "Sex")

## switch language
x$language <- "de"
x
x$tabulate()

## other interesting tables
od_table("OGD_veste309_Veste309_1")
od_table("OGD_konjunkturmonitor_KonMon_1")
od_table("OGD_krankenbewegungen_ex_LEISTUNGEN_1")
od_table("OGD_veste303_Veste203_1")
```

od_table_save *Saves/load opendata datasets via tar archives*

Description

`od_table_save()` creates a tar archive containing all relevant data from the OGD portal. `od_table_local()` parses the tar archive and recreates the `od_table` object.

Usage

```
od_table_save(x, file = NULL)

od_table_local(file)
```

Arguments

`x`	an object of class `od_table`
`file`	An archive file file for the dataset. For `od_table_save()`, the default is `{id}.tar.gz` where `id` denotes the OGD identifier.

Value

- for `od_table_save()`: the path to the generated file
- for `od_table_local()`: the OGD identifier

Examples

```
x <- od_table("OGD_krebs_ext_KREBS_1")

# save dataset as an archive
archive <- file.path(tempdir(), "table.tar.gz")
od_table_save(x, archive)

# read the saved archive
```

```
x2 <- od_table_local(archive)

# cleanup
file.remove(archive)
```

`other_endpoints` *Other endpoints of the STATcube REST API*

Description

Utilize the simple endpoints `/info` and `/table_rate_limit`. Those provide information about available locales and the amount of requests available for calls against the `/table` endpoint.

Usage

```
sc_info(language = c("en", "de"), key = NULL, server = "ext")

sc_rate_limit_table(language = c("en", "de"), key = NULL, server = "ext")

sc_rate_limit_schema(language = c("en", "de"), key = NULL, server = "ext")

sc_rate_limits(x)
```

Arguments

`language`	The language to be used for labeling. `"en"` or `"de"`
`key`	(`string`) An API key. To display your key, call `sc_browse_preferences()`.
`server`	A STATcube API server. Defaults to the external Server via `"ext"`. Other options are `"red"` for the editing server and `"prod"` for the production server. External users should always use the default option `"ext"`.
`x`	either a response-object (package `httr`), an object of class `sc_table` or an object of class `sc_schema`

Value

- `sc_info()`: a `data.frame` with two columns identifying possible languages
- `sc_rate_limit_table()`, `sc_rate_limit_schema()`, `sc_rate_limits()`: a `list` with elements
  - `remaining`: how much requests can be sent until the rate limit is reached
  - `limit`: the number of requests allowed per hour
  - `reset`: a timestamp when the rate limit will be reset
- `sc_rate_limits()`:

Functions

- `sc_info()`: returns information about all available database languages
- `sc_rate_limit_table()`: returns a `list` with information about current requests-limits with respect to the `/table` endpoint. It also shows when the limits reset which should be less than one hour after the current time.
- `sc_rate_limit_schema()`: returns a `list` with information about current requests-limits with respect to the `/schema` endpoint. It also shows when the limits reset which should be less than one hour after the current time.
- `sc_rate_limits()`: gets rate limits from response headers

Examples

```
sc_info()
sc_rate_limit_table()
sc_rate_limit_schema()
sc_rate_limits(sc_schema("str:group:deake005:X_B1"))
```

`sc_browse` *Links to important 'STATcube' and 'OGD' pages*

Description

A collection of links, to browse important 'STATcube' pages.

Usage

```
sc_browse(server = "ext")

sc_browse_preferences(server = "ext")

sc_browse_table(table, server = "ext")

sc_browse_database(database, server = NULL, open = FALSE)

sc_browse_catalogue(server = "ext")

sc_browse_ogd()
```

Arguments

`server`	A STATcube API server. Defaults to the external Server via `"ext"`. Other options are `"red"` for the editing server and `"prod"` for the production server. External users should always use the default option `"ext"`.
`table`	a table id
`database`	a database id
`open`	If `FALSE` (the default), open the infopage for the database. Otherwise, open the table view.

Value

the URL of a specific webpage which is opened by default in a web browser.

Functions

- `sc_browse()`: opens the home menu of 'STATcube'
- `sc_browse_preferences()`: opens the preference menu with the API key
- `sc_browse_table()`: shows the info page for a table
- `sc_browse_database()`: shows the info page for a database
- `sc_browse_catalogue()`: shows the data catalogue explorer
- `sc_browse_ogd()`: shows the landing page for OGD datasets

Examples

```
sc_browse()
sc_browse_preferences()
sc_browse_table('defaulttable_deake005')
sc_browse_database('deake005')
sc_browse_catalogue()
sc_browse_ogd()
```

`sc_cache` *Cache responses from the STATcube REST API*

Description

Functions to cache requested resources in the directory `~/.STATcubeR_cache` and reuse them in calls to `sc_table()`, `sc_table_custom()` `sc_schema()` and so forth. These functions are designed for testing and documentation and should not be regarded as part of the STATcubeR interface. The caching logic is likely to change in the future in which case `sc_cache_clear()` is required to purge old cache entries.

Usage

```
sc_cache_enable(verbose = TRUE)

sc_cache_disable()

sc_cache_enabled()

sc_cache_dir(dir = NULL)

sc_cache_files(x)

sc_cache_clear()
```

Arguments

`verbose`	print instructions on how to set up caching persistently via environment variables?
`dir`	a cache directory
`x`	an object of class `sc_table` or `sc_schema`

Details

Caching can be set up using environment variables. To set up a persistent cache for both Open Data and the REST API, the following lines in `.Renviron` can be used. The paths in this example are only applicable for UNIX-based operating systems.

```
STATCUBE_KEY_EXT   = YOUR_API_KEY_GOES_HERE
STATCUBE_CACHE     = TRUE
OD_CACHE_DIR       = "~/.cache/STATcubeR/open_data/"
STATCUBE_CACHE_DIR = "~/.cache/STATcubeR/api/"
```

If caching is enabled, there is no check to verify if the resources are unchanged in the server. Caching is not implemented for the endpoints `sc_info()` and `sc_rate_limit_table()`.

Value

- for `sc_cache_enable()`, `sc_cache_dir()`: the path to the cache-directory
- for `sc_cache_disable()`: `TRUE`
- for `sc_cache_enabled()`: `TRUE` if caching is enabled, `FALSE` otherwise
- for `sc_cache_files()`: the content of the cache associated with a file
- for `sc_cache_clear()`: `NULL`

Functions

- `sc_cache_enable()`: enables caching for the current R session
- `sc_cache_disable()`: disables caching for the current R session sc_cache_disable()
- `sc_cache_enabled()`: informs whether the cache is currently enabled
- `sc_cache_dir()`: get/set the directory used for caching
- `sc_cache_files()`: get the cache file associated with an object
- `sc_cache_clear()`: removes all files from the cache

sc_json_get_server *Get the server from a json request*

Description

parses a json request and returns a short string representing the corresponding STATcube server

Usage

```
sc_json_get_server(json)
```

Arguments

`json`	path to a request json

Value

`"ext"`, `"red"` or `"prod"` depending on the database uri in the json request

Examples

```
sc_json_get_server(sc_example('accomodation'))
```

sc_key *Manage your API Keys*

Description

Functions to get/set the STATcube API keys and make them available for calls against the STATcube API.

Usage

```
sc_key(server = "ext", test = FALSE)

sc_key_set(key, server = "ext", test = TRUE)

sc_key_get(server = "ext")

sc_key_prompt(server = "ext", test = TRUE)

sc_key_exists(server = "ext")

sc_key_valid(key = NULL, server = "ext")
```

Arguments

`server`	A STATcube API server. Defaults to the external Server via `"ext"`. Other options are `"red"` for the editing server and `"prod"` for the production server. External users should always use the default option `"ext"`.
`test`	Use `sc_key_valid()` to verify the key? If the key is invalid, an error is returned and the key will not be set or updated.
`key`	(`string`) An API key. To display your key, call `sc_browse_preferences()`.

Value

All functions return the key (invisibly) except for `sc_key_exists()` and `sc_key_valid()`, which return a `logical()` of length one.

Functions

- `sc_key()`: forwards to `sc_key_get()` if the key is already present. Otherwise, `sc_key_prompt()` will be invoked.
- `sc_key_set()`: can be used to pass the key as a parameter (`string`)
- `sc_key_get()`: returns the key, if it exists. Otherwise, an error is thrown.
- `sc_key_prompt()`: prompts for a key via `readline()`
- `sc_key_exists()`: returns `TRUE` if a key was set and `FALSE` otherwise.
- `sc_key_valid()`: performs a test request and returns `TRUE` if the key is valid and `FALSE` otherwise.

`sc_last_error` *Error handling for the STATcube REST API*

Description

In case API requests are unsuccessful, `STATcubeR` will throw errors to summarize the httr error type and its meaning. Requests are considered unsuccessful if one of the following applies

- The response returns `TRUE` for `httr::http_error()`.
- The response is not of type `"application/json"`

In some cases it is useful to get direct access to a faulty response object. For that purpose, it is possible to use `sc_last_error()` which will provide the httr response object for the last unsuccessful request.

Usage

```
sc_last_error()

sc_last_error_parsed()
```

Value

The return value from `httr::GET()` or `httr::POST()`.

Functions

- `sc_last_error_parsed()`: returns the last error as a list containing the response content and the response status

Examples

```
try(sc_table_saved("invalid_id"))
last_error <- sc_last_error()
httr::content(last_error)
str(sc_last_error_parsed())
```

`sc_recoder` *Recode sc_table objects*

Description

A collection of methods that can be used to modify an object of class sc_table by reference. Typical usage is to access the `recode` binding of an `sc_table` object and then use method chaining to perform recode operations.

```
x <- od_table("OGD_krebs_ext_KREBS_1")
x$recode$
  label_field("C-BERJ-0", "de", "JAHR")$
  label_measure("F-KRE", "de", "Anzahl")
```

See the example section for more details.

Methods

Public methods:

- `sc_recoder$new()`
- `sc_recoder$label_field()`
- `sc_recoder$label_measure()`
- `sc_recoder$level()`
- `sc_recoder$total_codes()`
- `sc_recoder$visible()`
- `sc_recoder$order()`

Method `new()`: Create a new recoder instance. This will automatically be performed during the setup of `sc_data` objects

Usage:

```
sc_recoder$new(x)
```

Arguments:

`x` the private environment of an `sc_data` object

Method `label_field()`**:** Change the label of a field variable

Usage:

```
sc_recoder$label_field(field, language, new)
```

Arguments:

`field` a field code

`language` a language, "de" or "en"

`new` the new label

Method `label_measure()`**:** Change the label of a measure variable

Usage:

```
sc_recoder$label_measure(measure, language, new)
```

Arguments:

`measure` a measure code

`language` a language "de" or "en"

`new` the new label

Method `level()`**:** Change the labels of a level

Usage:

```
sc_recoder$level(field, level, language, new)
```

Arguments:

`field` a field code

`level` a level code for the field

`language` a language "de" or "en"

`new` the new label for the level

Method `total_codes()`**:** Change the total code for a field

Usage:

```
sc_recoder$total_codes(field, new)
```

Arguments:

`field` a field code

`new` a level code for the field or `NA`. Will be used as the new total code. In case of `NA`, the total code will be unset.

Method `visible()`**:** set the visibility of a level. Invisible levels are omitted in the output of `$tabulate()` but don't affect aggregation

Usage:

```
sc_recoder$visible(field, level, new)
```

Arguments:

`field` a field code

`level` a level code for the field

`new` visibility. `TRUE` or `FALSE`

Method `order():` set the order of levels.

Usage:

`sc_recoder$order(field, new)`

Arguments:

`field` a field code

`new` the new order. A permutation of all level codes for the field. alternatively, an integer vector that defines the permutation.

Examples

```
x <- od_table("OGD_krebs_ext_KREBS_1")

x$recode$
  label_field("C-KRE_GESCHLECHT-0", "en", "SEX")$
  label_measure("F-KRE", "en", "NUMBER")$
  level("C-KRE_GESCHLECHT-0", "GESCHLECHT-1", "en", "MALE")

x$tabulate("C-KRE_GESCHLECHT-0", "F-KRE")

earnings <- od_table("OGD_veste309_Veste309_1")
earnings$recode$
  total_codes("C-A11-0", "A11-1")$
  total_codes("C-STAATS-0", "STAATS-9")$
  total_codes("C-VEBDL-0", "VEBDL-10")$
  total_codes("C-BESCHV-0", "BESCHV-1")

earnings$total_codes()

earnings$tabulate("C-STAATS-0")
earnings$recode$visible("C-STAATS-0", "STAATS-8", FALSE)
earnings$tabulate("C-STAATS-0")

earnings$recode$
  order("C-A11-0", c("A11-3", "A11-1", "A11-2"))
```

`sc_schema` *Create a request against the /schema endpoint*

Description

Invoke the **/schema** endpoint of the STATcube REST API. This endpoint can be used to get all available databases and tables as well as metadata about specific databases.

The main function sc_schema() can be used with any resource id. sc_schema_catalogue() and sc_schema_db() are very simple wrapper functions around sc_schema() and are comparable to the catalogue explorer or the table view of the STATcube GUI.

The responses of the API are tree-like data structures which are wrapped into a class called sc_schema to simplify the usage in R.

Usage

```
sc_schema(id = NULL, depth = NULL, language = NULL, key = NULL, server = "ext")

## S3 method for class 'sc_schema'
print(x, tree = NULL, ..., limit = 30)

sc_schema_flatten(x, type)

sc_schema_catalogue(depth = "folder", ...)

sc_schema_db(id, depth = "valueset", language = c("en", "de"), key = NULL)
```

Arguments

id	A resource identifier in uid format. In case of sc_schema_db(), this should be a database id. For sc_schema() any resource-id (folder, measure, table, ...) is accepted.
depth	If provided, the request will recurse into the given level. For datasets, available options are NULL (no recursion), "folder", "field" and "valueset". For the catalogue, only NULL and "folder" are applicable.
language	The language to be used for labeling. "en" or "de"
key	(string) An API key. To display your key, call sc_browse_preferences().
server	A STATcube API server. Defaults to the external Server via "ext". Other options are "red" for the editing server and "prod" for the production server. External users should always use the default option "ext".
x	an object of class sc_schema() i.e. the return value of sc_schema(), sc_schema_db() or sc_schema_catalogue().
tree	whether to use the data.tree package for printing.
limit, ...	passed to data.tree::print.Node() if tree is set to TRUE. Ignored otherwise.
type	a schema type such as "DATABASE", "VALUE" or "TABLE". See the API reference for a list of all schema types.

Value

- for sc_schema() and sc_schema_db(): an object of class sc_schema
- for sc_schema_flatten(): a data.frame
- for sc_schema_catalogue(): a list

Functions

- `sc_schema_flatten()`: turns a `sc_schema` object into a `data.frame`
- `sc_schema_catalogue()`: is similar to the catalogue explorer of the STATcube GUI and returns a tree-type object containing all databases and tables.
- `sc_schema_db()`: is similar to the table view of the STATcube GUI and gives information about all measures and classification fields for a specific database

Printing with data.tree

`limit` and `...` will simply be ignored if `tree` is set to `FALSE`, which is the default. The printing via `data.tree` can take longer than the default implementation because `x` will need to be converted into a `data.tree` node. To use `data.tree` printing permanently, use

```
options(STATcubeR.print_tree = TRUE)
```

Examples

```
my_catalogue <- sc_schema_catalogue()

## print
my_catalogue

## access the parsed catalogue
my_catalogue$Statistics$`Labour Market`
my_catalogue$Statistics$`Labour Market`$`Working hours (Labour Force Survey)`

db_schema <- sc_schema_db("deake005")

# printing
db_schema

# access child nodes
db_schema$`Demographic Characteristics`
db_schema$`Demographic Characteristics`$Gender$Gender
db_schema$`Demographic Characteristics`$Gender$Gender$male

# access the raw response from httr::GET()
my_response <- attr(db_schema, "response")
my_response$headers$date
my_content <- httr::content(my_response)
my_content$label

# print with data.tree

 treeX_B1 <- sc_schema("str:group:deake005:X_B1", depth = "valueset")
 print(treeX_B1, tree = TRUE)
```

`sc_table` *Create a request against the /table endpoint*

Description

Send requests against the `/table` endpoint of the STATcube REST API. The requests can use three formats with corresponding functions

- `sc_table()` uses a json file downloaded via the STATcube GUI
- `sc_table_custom()` uses the ids of a database, measures and fields
- `sc_table_saved()` uses a table uri of a saved table.

Those three functions all return an object of class `"sc_table"`.

Usage

```
sc_table(json, language = NULL, add_totals = TRUE, key = NULL, json_file = NA)

sc_examples_list()

sc_example(filename)

sc_table_saved_list(key = NULL, server = "ext")

sc_table_saved(table_uri, language = NULL, key = NULL, server = "ext")
```

Arguments

`json`	Path to a json file, which was downloaded via the STATcube GUI ("Open Data API Request"). Alternatively, a json string which passes `jsonlite::validate()`.
`language`	The language to be used for labeling. `"en"` (the default) will use english. `"de"` uses German. The third option `"both"` will import both languages by sending two requests to the `/table` endpoint.
`add_totals`	Should totals be added for each classification field in the json request?
`key`	(`string`) An API key. To display your key, call `sc_browse_preferences()`.
`json_file`	Deprecated. Use `json` instead
`filename`	The name of an example json file.
`server`	A STATcube API server. Defaults to the external Server via `"ext"`. Other options are `"red"` for the editing server and `"prod"` for the production server. External users should always use the default option `"ext"`.
`table_uri`	Identifier of a saved table as returned by `sc_table_saved_list()`

Value

An object of class `sc_table` which contains the return value of the `httr::POST()` request in `obj$response`. The object also provides member functions to parse this response object. See sc_table_class for the class documentation.

Examples

```
my_table <- sc_table(json = sc_example("population_timeseries.json"))

# print
my_table

# get matadata for the table
my_table$meta

# get a data.frame
as.data.frame(my_table)

# get metadata for field 2
my_table$field(2)


# get the ids and labels of all saved tables
(saved_tables <- sc_table_saved_list())
table_uri <- saved_tables$id[1]

# get a table based on one of these ids
my_response <- sc_table_saved(table_uri)
as.data.frame(my_response)
```

`sc_table_custom` *Create custom tables*

Description

Define requests against the /table endpoint by providing URIs to databases, measures and fields. The URIs can be obtained using `sc_schema_db()`. See the Custom tables article for more details.

Usage

```
sc_table_custom(
  db,
  measures = c(),
  dimensions = c(),
  language = c("en", "de"),
  add_totals = TRUE,
  key = NULL,
  recodes = NULL,
  dry_run = FALSE
)

sc_recode(field, map = NULL, total = FALSE)
```

Arguments

`db`	The uid of a database. Must be of type `DATASET`
`measures`	A character vector of uids for measures. Each entry must be of type `MEASURE`, `STAT_FUNCTION` or `COUNT`.
`dimensions`	A character vector of dimensions for the cube. Can be either of type `FIELD` or type `VALUESET`. Those entries are referred to as `fields` in the parsed API response
`language`	The language to be used for labeling. "en" (the default) will use English. "de" uses German.
`add_totals`	Should totals be added for each classification field in the json request? Ignored if `recodes` is used.
`key`	(`string`) An API key. To display your key, call `sc_browse_preferences()`.
`recodes`	One or more recodes that were generated via `sc_recode()`. If more than one recode is supplied, recodes should be concatenated with `c()`.
`dry_run`	If `TRUE`, no request is sent to the API. Instead, type checks are performed and the json request is returned as a string. Defaults to `FALSE`.
`field`	An uid of a classification field to be recoded. The provided uid should also be passed in the `dimensions` parameter of `sc_table_custom()`.
`map`	A list of ids for values (type `VALUE`) This can also be a nested list if items should be grouped. See examples
`total`	Add totals to the field? If `map` is provided, the totals will correspond to the filtered data.

Value

- for `sc_table_custom()`: an object of class `sc_table`
- for `sc_recode()`: a `list` that is a suitable input for parameter `"recode"` in `sc_table_custom()`

Functions

- `sc_recode()`: creates a recode object which can be used for the `recode` parameter of `sc_table_custom()`

Schema objects in parameters

it is possible to pass `sc_schema` objects (usually generated by `sc_schema_db()`) instead of ids in `sc_table_custom()` and `sc_recode()`. If provided, the schema objects will be converted into ids via `$id`.

Error handling

Unfortunately, the API gives fairly vague error messages in case a custom table request is ill defined. For this reason, `sc_table_custom()` applies some simple heuristics and throws warnings if inconsistencies in the provided parameters are recognized. The following conditions are currently checked

- the parameter db is of type `DATABASE`

- all entries in `measures` are of type `MEASURE`, `COUNT` or `STATFN`
- all entries in `dimensions` are of type `VALUESET` or `FIELD`
- all entries in `field` are of type `VALUESET` or `FIELD`
- all entries in `map` are of type `VALUE`
- all fields in `recodes` are also present in `dimensions`
- the first two arguments of `sc_recode()` are consistent, i.e. if the provided `VALUE`s belong to the `VALUESET/FIELD`

Examples

```
sc_table_custom("str:database:detouextregsai")

sc_table_custom(
  "str:database:detouextregsai",
  dimensions = "str:field:detouextregsai:F-DATA1:C-SDB_TIT-0"
)

sc_table_custom(
  db = "str:database:detouextregsai",
  measures = c(
    "str:statfn:detouextregsai:F-DATA1:F-ANK:SUM",
    "str:measure:detouextregsai:F-DATA1:F-UEB"
  ),
  dimensions = c(
    "str:field:detouextregsai:F-DATA1:C-SDB_TIT-0",
    "str:valueset:detouextregsai:F-DATA1:C-C93-2:C-C93SUM-0"
  )
)

schema <- sc_schema_db("detouextregsai")
region <- schema$`Other Classifications`$`Tourism commune [ABO]`$
  `Regionale Gliederung (Ebene +1)`
month <- schema$`Mandatory fields`$`Season/Tourism Month`

x <- sc_table_custom(
  schema,
  schema$Facts$Arrivals,
  list(month, region),
  recodes = c(
    sc_recode(region, total = FALSE, map = list(
      region$Achensee,
      list(region$Arlberg, region$`Ausseerland-Salzkammergut`)
    )),
    sc_recode(month, total = FALSE)
  )
)
x$tabulate()
```

sc_tabulate *Turn sc_data objects into tidy data frames*

Description

`sc_tabulate()` extracts the data in the table and turns it into a tidy data.frame. It applies labeling of the data and transforms time variables into a `Date` format if they satisfy certain 'STATcube' standards.

`sc_tabulate(table, ...)` is just an alias for `table$tabulate(...)` and was added so this rather complicated method can have a separate documentation page. It is recommended to use the `table$tabulate()` syntax

the `...` argument decides which measures and/or fields should be included in the output. If no measures are given, all measures are included. The same is true for fields.

Usage

```
sc_tabulate(
  table,
  ...,
  .list = NULL,
  raw = FALSE,
  parse_time = TRUE,
  recode_zeros = inherits(table, "sc_table"),
  language = NULL,
  sort = FALSE
)
```

Arguments

`table`	An object of class `sc_data`
`...`	Names of measures and/or fields
`.list`	allows to define the arguments for `...` as a character vector.
`raw`	If FALSE (the default), apply labeling to the dataset. Otherwise, return codes.
`parse_time`	Should time variables be converted into a `Date` format? Ignored if `raw` is set to `TRUE`.
`recode_zeros`	turn zero values into `NA`s
`language`	The language to be used for labeling. By default, the dataset language (`table$language`) is used.
`sort`	If `TRUE`, the resulting data will be sorted by all provided field values

Details

Aggregation is done as follows

- First, all columns that provide a total code via `table$total_codes()` will be used to filter for `column` == `total_code` or `column` != `total_code`
- Then, the remaining data is aggregated using `rowsum()`

The ellipsis (`...`) supports partial matching of codes and labels. See Examples

For objects of class `sc_table` two additional operations are performed.

- zeros are recoded to `NA`s
- rounding is done according to the precision of each measure. Rounding happens after the recoding to `NA` values

Value

a `data.frame`

See Also

sc_table_class

Examples

```
############################ OGD Data ########################################

table <- od_table("OGD_veste309_Veste309_1")

# no arguments -> same output as `table$data`
table$tabulate()

# provide some fields -> aggregate to keep only these fields
table$tabulate("Sex", "Citizenship")

# provide some measures -> drop all other measures from the output
table$tabulate("Arithmetic mean")

# mixture of measures and fields  -> keep exactly those columns
table$tabulate("Sex", "Arithmetic mean")

## define total codes
table$total_codes(
  `C-A11-0` = "A11-1",
  `C-STAATS-0` = "STAATS-9",
  `C-VEBDL-0` = "VEBDL-10",
  `C-BESCHV-0` = "BESCHV-1"
)

## alternatively, use partial matching to define totals
table$total_codes(
  Sex = "Sum total",
  Citizenship = "Total",
  Region = "Total",
  `Form of employment` = "Total"
)

# filter for totals in `Region (NUTS2)` and `Form of employment`. Drop totals
# in `Sex` and `Citizenship`.
table$tabulate("Sex", "Citizenship")

## switch language
table$language <- "de"

## `...` matches for codes and labels
table$tabulate("C-A11-0", "Staats", "2. Quartil (Median)")

## Keep totals in the output by removing total codes
table$tabulate("C-A11-0")      # -> 2 rows: "male" "female"
table$total_codes(`C-A11-0` = NA)
table$tabulate("C-A11-0")      # -> 3 rows: "total", "male", "female"

## table$tabulate(...) is an alias for sc_tabulate(table, ...)
sc_tabulate(table, "C-A11-0")

######################## 'STATcube' REST API ################################


table_tourism <- sc_table(sc_example("accomodation.json"), "de")

table_tourism$tabulate()
table_tourism$tabulate("Saison/Tourismusmonat")
table_tourism$tabulate("Saison/Tourismusmonat", "Ankünfte")
table_tourism$tabulate("Ankünfte")
```

Index