

# Package ‘FAOSTAT’

July 21, 2025

**Type** Package

**Title** Download Data from the FAOSTAT Database

**Version** 2.4.0

**Date** 2024-05-20

**Author** Michael C. J., Markus Gesmann, Filippo Gheri, Paul Rougieux <paul.rougieux@gmail.com>, Sebastian Campbell

**Maintainer** Paul Rougieux <paul.rougieux@gmail.com>

**Description** Download Data from the FAOSTAT Database of the Food and Agricultural Organization (FAO) of the United Nations.

A list of functions to download statistics from FAOSTAT (database of the FAO <<https://www.fao.org/faostat/>>) and WDI (database of the World Bank <<https://data.worldbank.org/>>), and to perform some harmonization operations.

**URL** <https://gitlab.com/paulrougieux/faostatpackage>

**BugReports** <https://gitlab.com/paulrougieux/faostatpackage/-/issues>

**Imports** RJSONIO (>= 0.96-0), plyr (>= 1.7.1), data.table (>= 1.8.2), MASS (>= 7.3-22), classInt (>= 0.1-19), labeling (>= 0.1), httr (>= 1.0)

**License** GPL (>= 2)

**Encoding** UTF-8

**LazyData** yes

**ZipData** no

**VignetteBuilder** knitr

**Suggests** knitr, testthat

**RoxygenNote** 7.3.1

**NeedsCompilation** no

**Repository** CRAN

**Date/Publication** 2024-05-22 12:40:02 UTC

## Contents

FAOSTAT-package	2
Aggregation	3
change_case	4
chConstruct	5
check_country_overlap	6
chgr	6
CHMT	7
constructSYB	8
download_faostat_bulk	9
ebind	11
FAOcountryProfile	11
FAOmetaTable	11
FAOregionProfile	11
fillCountryCode	12
geogr	12
getWDI	13
getWDImetaData	14
getWDItoSYB	15
grConstruct	16
indConstruct	17
lsgr	17
mergeSYB	18
overlap	19
printLab	19
read_dataset_dimension	20
read_fao	20
scaleUnit	22
search_dataset	23
shConstruct	24
translateUnit	25
translate_countrycodes	25
<b>Index</b>	<b>27</b>

---

FAOSTAT-package	<i>A complementary package to the FAOSTAT database and the Statistical Yearbook of the Food and Agricultural Organization of the United Nations.</i>
-----------------	--

---

## Description

Download Data from the FAOSTAT Database of the Food and Agricultural Organization (FAO) of the United Nations. A list of functions to download statistics from FAOSTAT (database of the FAO <https://www.fao.org/faostat/>) and WDI (database of the World Bank <https://data.worldbank.org/>), and to perform some harmonization operations.

**See Also**

Useful links:

- <https://gitlab.com/paulrougieux/faostatpackage>
- Report bugs at <https://gitlab.com/paulrougieux/faostatpackage/-/issues>

---

 Aggregation

*Compute Aggregates*


---

**Description**

The function takes a relational data frame and computes the aggregation based on the relation specified.

**Usage**

```
Aggregation(
  data,
  aggVar,
  weightVar = rep(NA, length(aggVar)),
  year = "Year",
  relationDF = FAOcountryProfile[, c("FAOST_CODE", "M49_FAOST_CODE")],
  aggMethod = rep("sum", length(aggVar)),
  applyRules = TRUE,
  keepUnspecified = TRUE,
  unspecifiedCode = 0,
  thresholdProp = rep(0.65, length(aggVar))
)
```

**Arguments**

<code>data</code>	The data frame containing the country level data.
<code>aggVar</code>	The vector of names of the variables to be aggregated.
<code>weightVar</code>	The vector of names of the variables to be used as weighting when the aggregation method is weighted.
<code>year</code>	The column containing the time information.
<code>relationDF</code>	A relational data frame which specifies the territory and the mother country. At least one column must have a correspondent variable name in the dataset.
<code>aggMethod</code>	Can be a single method for all data or a vector specifying different method for each variable. The method can be "sum", "mean", "weighted.mean".
<code>applyRules</code>	Logical, specifies whether the thresholdProp rule must be applied or not.
<code>keepUnspecified</code>	Whether countries with unspecified region should be aggregated into an "Unspecified" group or simply drop. Default to create the new group.

unspecifiedCode      The output code of the unspecified group.

thresholdProp      The vector of the missing threshold for the aggregation rule to be applied. The default is set to only compute aggregation if there are more than 65 percent of data available (0.65).

### Details

The length of `aggVar`, `aggMethod`, `weightVar`, `thresholdProp` must be the same.

Aggregation should not be computed if insufficient countries have reported data. This corresponds to the argument `thresholdProp` which specifies the percentage which of country must report data (both for the variable to be aggregated and the weighting variable).

### Examples

```
## example.df = data.frame(FAOST_CODE = rep(c(1, 2, 3), 2),
##                          Year = rep(c(2010, 2011), c(3, 3)),
##                          value = rep(c(1, 2, 3), 2),
##                          weight = rep(c(0.3, 0.7, 1), 2))

## Lets aggregate country 1 and 2 into one country and keep country
## 3 separate.
## relation.df = data.frame(FAOST_CODE = 1:3, NEW_CODE = c(1, 1, 2))
```

---

change_case	<i>Change case of column names</i>
-------------	------------------------------------

---

### Description

Columns from FAOSTAT frequently have parentheses and other non-alphanumeric characters. This suite of functions seeks to give control over these names for easier data analysis

### Usage

```
change_case(
  old_names,
  new_case = c("make.names", "unsanitised", "unsanitized", "snake_case"),
  ...
)
```

### Arguments

`old_names`      character. Vector of the names to be changed

`new_case`      character. Choice of new names:

- `make_names` - (default) use the [make.names](#) function in R to sanitise names
- `unsanitised/unsanitized` - Return names as they are

- snake\_case - Names are converted to lowercase and separators are replaced with underscores
- ... extra arguments to pass to sanitisation function (only works for [make.names](#))

---

chConstruct                      *Construct year to year change*

---

### Description

A function for constructing year to year change

### Usage

```
chConstruct(
  data,
  origVar,
  country = "FAOST_CODE",
  year = "Year",
  newVarName = NA,
  n = 1
)
```

### Arguments

data	The data frame containing the data
origVar	The variable in which the year to year change is to be calculated
country	The column representing the index of country.
year	The column represing the index of year.
newVarName	The name assigned to the new variable, if missing then .CH will be appended.
n	The period for the change rate to be calculated.

### Value

A data frame containing the computed year to year change rate.

---

check\_country\_overlap *This function perform some check on the data*

---

### Description

The function only works for FAOST\_CODE. If the country coding system is not in FAOST\_CODE then use the translateCountryCode function to translate it.

### Usage

```
check_country_overlap(
  var,
  year = "Year",
  data,
  type = c("overlap", "multiChina"),
  take = c("simpleCheck", "takeNew", "takeOld", "complete")
)
```

### Arguments

var	The variable to be checked.
year	The column which index the time.
data	The data frame.
type	The type of check.
take	The type of check/replacement to be done in case of type equals to overlap.

### Examples

```
test.df =
  data.frame(FAOST_CODE = rep(c(51,167,199), each = 3),
    Year = rep(c(1990:1992), 3),
    Value = c(c(3,4,4), c(2,2,2), c(1,2,NA)))
check_country_overlap(var = "Value", data = test.df, type = "overlap", take = "simpleCheck")
check_country_overlap(var = "Value", data = test.df, type = "overlap", take = "takeNew")
check_country_overlap(var = "Value", data = test.df, type = "overlap", take = "takeOld")
check_country_overlap(var = "Value", data = test.df, type = "overlap", take = "complete")
```

---

chgr

*Absolute change between the year*

---

### Description

Function for generating the n-period absolute change

**Usage**

```
chgr(x, n = 1)
```

**Arguments**

x	The time series for the change to be calculated.
n	The period for the growth to be calculated over.

**Details**

In order to ensure the change calculated is reliable, the following rule are applied.

1. 50% of the data must be present.
2. The length of the time series must be greater than n

Otherwise the growth will not be computed.

**Value**

The n-period change of the time series.

**Examples**

```
test.ts = abs(rnorm(100))
chgr(test.ts, 1)
chgr(test.ts, 3)
chgr(test.ts, 10)
```

---

CHMT

*This function avoids double counting of China.*

---

**Description**

This function should only be used when performing aggregations.

**Usage**

```
CHMT(var, data, year = "Year")
```

**Arguments**

var	The variables that require to be sanitized.
data	The data frame which contains the data
year	The column which correspond to the year.

## Details

We decide to use the smaller subsets in the regional level because weighting variable may not exist for other variables for the larger subsets.

The function only work for FAOST\_CODE, if the country coding system is not in FAOST\_CODE then use the translateCountryCode function to translate it.

---

constructSYB	<i>Construct/Creat new variable.</i>
--------------	--------------------------------------

---

## Description

A function used to construct new variables from existing variables.

## Usage

```
constructSYB(
  data,
  origVar1,
  origVar2,
  newVarName = NA,
  constructType = c("share", "growth", "change", "index"),
  grFreq = 1,
  grType = c("ls", "geo"),
  baseYear = 2000
)
```

## Arguments

data	The data frame containing the raw variable
origVar1	The variable name to be used in construction, refer to Details for more information and useage.
origVar2	The variable name to be used in construction, refer to Details for more information and useage.
newVarName	The name assigned to the new variable, if missing then .SC/.SH/.GR/.CH will be appended depending on the type of construction
constructType	The type of construction, refer to Details for more information.
grFreq	The frequency for the growth rate to be computed.
grType	The method for the growth to be calculated, currently supports least squares and geometric.
baseYear	The base year to be used for constructing index.

## Details

Currently two types of construction are supported, either share or growth rate computation.

Share can be a share of total or share of another variable depending on whether an additional variable is supplied or not.



**Value**

A data frame containing both the original data frame and the processed data and also a list indicating whether the construction passed or failed.

---

download\_faostat\_bulk *Download bulk data from the faostat website*  
<https://www.fao.org/faostat/en/#data>

---

**Description**

- `get_faostat_bulk()` loads the given data set code and returns a data frame.
- `download_faostat_bulk()` loads data from the given url and saves it to a compressed zip file.
- `read_faostat_bulk()` Reads the compressed .csv .zip file into a data frame. More precisely it unzips the archive. Reads the main csv file within the archive. The main file has the same name as the name of the archive. Note: the zip archive might also contain metadata files about Flags and Symbols.

In general you should load the data with the function `get_faostat_bulk()` and a dataset code. The other functions are lower level functions that you can use as an alternative. You can also explore the datasets and find their download URLs on the FAOSTAT website. Explore the website to find out the data you are interested in <https://www.fao.org/faostat/en/#data> Copy a "bulk download" url, for example they are located in the right menu on the "crops" page <https://www.fao.org/faostat/en/#data/QC> Note that faostat bulk files with names ending with "normalized" are in long format with a year column instead of one column for each year. The long format is preferable for data analysis and this is the format returned by the `get_faostat_bulk()` function.

**Usage**

```
download_faostat_bulk(url_bulk, data_folder = ".")

read_faostat_bulk(zip_file_name, encoding = "latin1", rename_element = TRUE)

get_faostat_bulk(code, data_folder = tempdir(), subset = "All Data Normalized")

read_bulk_metadata(dataset_code)
```

**Arguments**

<code>url_bulk</code>	character url of the faostat bulk zip file to download
<code>data_folder</code>	character path of the local folder where to download the data
<code>zip_file_name</code>	character name of the zip file to read
<code>encoding</code>	parameter passed to 'read.csv'.
<code>rename_element</code>	boolean Rename the element column to snake case. To facilitate the use of elements as column names later when the data frame gets reshaped to a wider format. Replace non alphanumeric characters by underscores.

code	character. Dataset code
subset	character. Use read_bulk_metadata. Request all data, normalised data or region
dataset_code	character. Dataset code

**Value**

data frame of FAOSTAT data

data frame of FAOSTAT data

data frame of FAOSTAT data

**Author(s)**

Paul Rougieux

**Examples**

```
## Not run:

# Create a folder to store the data
data_folder <- "data_raw"
dir.create(data_folder)

# Load crop production data
crop_production <- get_faostat_bulk(code = "QCL", data_folder = data_folder)

# Cache the file i.e. save the data frame in the serialized RDS format for faster load time later.
saveRDS(crop_production, "data_raw/crop_production_e_all_data.rds")
# Now you can load your local version of the data from the RDS file
crop_production <- readRDS("data_raw/crop_production_e_all_data.rds")

# Use the lower level functions to download zip files,
# then read the zip files in separate function calls.
# In this example, to avoid a warning about "examples lines wider than 100 characters"
# the url is split in two parts: a common part 'url_bulk_site' and a .zip file name part.
# In practice you can enter the full url directly as the `url_bulk` argument.
# Notice also that I have choosen to load global data in long format (normalized).
url_bulk_site <- "https://fenixservices.fao.org/faostat/static/bulkdownloads"
url_crops <- file.path(url_bulk_site, "crop_production_E_All_Data_(Normalized).zip")
url_forestry <- file.path(url_bulk_site, "Forestry_E_All_Data_(Normalized).zip")
# Download the files
download_faostat_bulk(url_bulk = url_forestry, data_folder = data_folder)
download_faostat_bulk(url_bulk = url_crops, data_folder = data_folder)

# Read the files and assign them to data frames
crop_production <- read_faostat_bulk("data_raw/crop_production_E_All_Data_(Normalized).zip")
forestry <- read_faostat_bulk("data_raw/Forestry_E_All_Data_(Normalized).zip")

# Save the data frame in the serialized RDS format for fast reuse later.
saveRDS(crop_production, "data_raw/crop_production_e_all_data.rds")
```

```
saveRDS(forestry, "data_raw/forestry_e_all_data.rds")
## End(Not run)
```

*ebind*                      *A function to bind the different entity level.*

**Description**

A data frame is chosen over the list is solely for the purpose of transition to ggplot2.

**Usage**

```
ebind(territory = NULL, subregion = NULL, region = NULL, world = NULL)
```

**Arguments**

territory	The data frame which contains the territory/country level data
subregion	The sub aggregated region aggregate
region	The macro region aggregate
world	The world aggregate

FAOcountryProfile              *Country profile*

**Description**

The country profile containing the codes and names of countries.

FAOmetaTable                      *The search tree for FAOSTAT3*

**Description**

A table containing the relationship between the domain, element, item codes for downloading data from the FAOSTAT API.

FAOregionProfile                  *Regional profile*

**Description**

Region profile containing the codes, names and regional classifications of countries.

---

fillCountryCode      *A function to get country code when not available in data.*

---

### Description

This function can be useful when a dataset provided does not have a country code available.

### Usage

```
fillCountryCode(country, data, outCode = "FAOST_CODE")
```

### Arguments

country	The column name of the data which contains the country name
data	The data frame to be matched
outCode	The output country code system, defaulted to FAO standard.

---

geogr      *Geometric growth rate*

---

### Description

Function for generating the n-period rolling geometric growth rate.

### Usage

```
geogr(x, n = 1)
```

### Arguments

x	The time series for the growth rate to be calculated.
n	The period for the growth to be calculated over.

### Details

In order to ensure the growth rate calculated is reliable, the following rule are applied.

1. 50% of the data must be present.
2. The length of the time series must be greater than n

Otherwise the growth will not be computed.

### Value

The n-period geometric growth rate of the time series.

## Examples

```
test.ts = abs(rnorm(100))
geogr(test.ts, 1)
geogr(test.ts, 3)
geogr(test.ts, 10)
```

---

getWDI

*Access to World Bank WDI API*

---

## Description

A function to extract data from the World Bank API

Please refer to <https://data.worldbank.org/> for any difference between the country code system. Further details on World Bank classification and methodology are available on that website.

## Usage

```
getWDI(  
  indicator = "SP.POP.TOTL",  
  name = NULL,  
  startDate = 1960,  
  endDate = format(Sys.Date(), "%Y"),  
  printURL = FALSE,  
  outputFormat = "wide"  
)
```

## Arguments

indicator	The World Bank official indicator name.
name	The new name to be used in the column.
startDate	The start date for the data to begin
endDate	The end date.
printURL	Whether the url link for the data should be printed
outputFormat	The format of the data, can be 'long' or 'wide'.

## Details

Sometime after 2016, there was a change in the api according to <https://datahelpdesk.worldbank.org/knowledgebase/articles/889392-about-the-indicators-api-documentation> "Version 2 (V2) of the Indicators API has been released and replaces V1 of the API. V1 API calls will no longer be supported. To use the V2 API, you must place v2 in the call.

Original (2011) source by Markus Gesmann: <https://lamages.blogspot.it/2011/09/setting-initial-view-of-mot.html> Also available at <https://www.magesblog.com/post/2011-09-25-accessing-and-plotting-world-bank-data>

**Value**

A data frame containing the desired World Bank Indicator

**See Also**

[getFAO](#), [getWDItoSYB](#)

and the WBI package <https://cran.r-project.org/package=WDI> for an implementation with many more features.

**Examples**

```
## pop.df = getWDI()
```

---

getWDImetaData	<i>World Bank Indicator Metadata</i>
----------------	--------------------------------------

---

**Description**

A function to extract the definition and the meta data from the World Bank API

**Usage**

```
getWDImetaData(  
  indicator,  
  printMetaData = FALSE,  
  saveMetaData = FALSE,  
  saveName = "worldBankMetaData"  
)
```

**Arguments**

indicator	The World Bank official indicator name.
printMetaData	logical, print out the meta data information
saveMetaData	logical, whether meta data should be saved as a local csv file.
saveName	The name of the file for the meta data to save to.

**Examples**

```
## pop.df = getWDImetaData("SP.POP.TOTL",  
##                           printMetaData = TRUE, saveMetaData = TRUE)
```

---

`getWDItoSYB`*Access to World Bank WDI API*

---

### Description

The function downloads data from the World Bank API.

### Usage

```
getWDItoSYB(  
  indicator = "SP.POP.0014.TO.ZS",  
  name = NULL,  
  startDate = 1960,  
  endDate = format(Sys.Date(), "%Y"),  
  printURL = FALSE,  
  getMetaData = TRUE,  
  printMetaData = FALSE,  
  saveMetaData = FALSE,  
  outputFormat = c("wide", "long")  
)
```

### Arguments

<code>indicator</code>	The World Bank official indicator name.
<code>name</code>	The new name to be used in the column.
<code>startDate</code>	The start date for the data to begin
<code>endDate</code>	The end date.
<code>printURL</code>	Whether the url link for the data should be printed
<code>getMetaData</code>	Whether the data definition and the meta data should be downloaded as well.
<code>printMetaData</code>	logical, print out the meta data information
<code>saveMetaData</code>	logical, whether meta data should be saved as a local csv file
<code>outputFormat</code>	The format of the data, can be 'long' or 'wide'.

### Value

A list containing the following elements

**data** The country level data

**aggregates** The aggregates provided by the World Bank

**metaData** The metaData associated with the data

**results** The status of the download, whether success/failed

### See Also

[getWDI](#), [getFAO](#)

**Examples**

```
## pop.df = getWDItoSYB(name = "total_population",
##                       indicator = "SP.POP.TOTL")
```

---

grConstruct

*Construct Growth rate*


---

**Description**

A function for constructing growth rate variables.

**Usage**

```
grConstruct(data, origVar, newVarName = NA, type = c("geo", "ls", "ch"), n = 1)
```

**Arguments**

data	The data frame containing the data
origVar	The variable in which the growth is to be calculated
newVarName	The name assigned to the new variable, if missing then <i>.SC/.SH/.GR</i> will be appended depending on the type of construction.
type	The type of growth rate, can be least squares or geometric
n	The period for the growth rate to be calculated (Refer to the <i>lsgr</i> or the <i>geogr</i> functions.)

**Value**

A data frame containing the computed growth rate.

**Examples**

```
test.df2 = data.frame(FAOST_CODE = rep(c(1, 5000), each = 5),
                      Year = rep(1990:1994, 2),
                      a = rep(1:5, 2), b = rep(1:5, 2))
grConstruct(test.df2, origVar = "a", type = "geo", n = 1)
grConstruct(test.df2, origVar = "a", type = "geo", n = 3)
grConstruct(test.df2, origVar = "a", type = "geo", n = 5)
```



---

indConstruct	<i>Construct indices</i>
--------------	--------------------------

---

**Description**

A function for constructing indices

**Usage**

```
indConstruct(data, origVar, newVarName = NA, baseYear = 2000)
```

**Arguments**

data	The data frame containing the data
origVar	The variable in which the indices is to be computed
newVarName	The name assigned to the new variable, if missing then .SC/.SH/.GR/.CH/.IND will be appended depending on the type of construction.
baseYear	The year which will serve as the base

**Value**

The indice

**Examples**

```
test.df = data.frame(FAOST_CODE = rep(1, 100), Year = 1901:2000,
                    test = 1:100)
indConstruct(test.df, origVar = "test", baseYear = 1950)
```

---

lsgr	<i>Least squares growth rate</i>
------	----------------------------------

---

**Description**

Function for generating the n-period rolling least squares growth rate.

**Usage**

```
lsgr(x, n = 1)
```

**Arguments**

x	The time series for the growth rate to be calculated
n	The period for the growth to be calculated over.

**Details**

Missing values are omitted in the regression. (Will need to check this.)

WONTFIX (Michael): There is still some error associated with this function, will need to investigate further. Will need a rule for this, when the fluctuation is large and data are sufficient then take the lsgr, otherwise the geogr.

In order to ensure the growth rate calculated is reliable, the following rule are applied.

1. 50% of the data must be present.
2. The length of the time series must be greater than n.

Otherwise the growth will not be computed.

**Value**

The n-period least squares growth rate of the time series

**Examples**

```
test.ts = abs(rnorm(100))
lsgr(test.ts, 1)
lsgr(test.ts, 3)
lsgr(test.ts, 10)
```

---

mergeSYB

---

*Function for merging data from different source.*


---

**Description**

This function searches for supported country system and translate the data to allow for join.

**Usage**

```
mergeSYB(x, y, outCode = "FAOST_CODE", all = TRUE, ...)
```

**Arguments**

x	data frames, or objects to be coerced to one.
y	data frames, or objects to be coerced to one.
outCode	The country code system to be used to join the different sources.
all	Same as the merge function, defaulted to an outer join.
...	Arguments to be passed on to the merge function.

**Details**

The names of the data to be merged has to be the same as the FAOCountryProfile code name.

---

overlap	<i>This function checks whether there are overlapping between the transitional countries.</i>
---------	---

---

**Description**

This function checks whether there are overlapping between the transitional countries.

**Usage**

```
overlap(old, new, var, year = "Year", data, take)
```

**Arguments**

old	The FAOST_CODE of the old countries
new	The FAOST_CODE of the new countries
var	The variable to be checked
year	The column which index the time.
data	The data frame
take	The type of check/replacement to be done.

---

printLab	<i>Print labels</i>
----------	---------------------

---

**Description**

A function to print standardised formatted labels without having messy codes in the functions.

**Usage**

```
printLab(label, span = FALSE, width = getOption("width"))
```

**Arguments**

label	The label to be printed
span	Whether the dash should span the whole width of the screen(80 characters)
width	The width of the screen.

**Value**

The formatted print

---

```
read_dataset_dimension
```

*Examine dimensions of a dataset*

---

### Description

Lists the dimensions of a dataset including ids and labels. These can be used to query dataset dimension names and the codes therein. They can also be used to access groups, flags, units and the glossary

### Usage

```
read_dataset_dimension(dataset_code)
```

```
read_dimension_metadata(dataset_code, dimension_code)
```

### Arguments

`dataset_code` character. Dataset as obtained from the code column of [search\\_dataset](#)

`dimension_code` character. Dimensions as obtained from `read_dataset_dimensions`

---

```
read_fao
```

*Access FAOSTAT API*

---

### Description

Uses the same functionality as the web interface to pull data from the FAOSTAT API. Contains most of its parameters. Currently only works for datasets that have area, item, element and year. Values for Chinese countries are not yet deduplicated.

### Usage

```
read_fao(
  area_codes,
  element_codes,
  item_codes,
  year_codes,
  area_format = c("M49", "FAO", "ISO2", "ISO3"),
  item_format = c("CPC", "FAO"),
  dataset = "RL",
  metadata_cols = c("codes", "units", "flags", "notes"),
  clean_format = c("make.names", "unsanitised", "unsanitized", "snake_case"),
  include_na = FALSE,
  language = c("en", "fr", "es")
)
```

```

getFAO(
  area_codes,
  element_codes,
  item_codes,
  year_codes,
  area_format = c("M49", "FAO", "ISO2", "ISO3"),
  item_format = c("CPC", "FAO"),
  dataset = "RL",
  metadata_cols = c("codes", "units", "flags", "notes"),
  clean_format = c("make.names", "unsanitised", "unsanitized", "snake_case"),
  include_na = FALSE,
  language = c("en", "fr", "es")
)

```

### Arguments

area_codes	character. FAOSTAT area codes
element_codes	character. FAOSTAT element codes
item_codes	character. FAOSTAT item codes
year_codes	character. Vector of desired years
area_format	character. Desired area code type in output (input still needs to use FAOSTAT codes)
item_format	character. Item code
dataset	character. FAO dataset desired, e.g. RL, FBS
metadata_cols	character. Metadata columns to include in output
clean_format	character/function. Whether to clean columns. Either one of the formats described in [change_case] or a formatting function
include_na	logical. Whether to include NAs for combinations of dimensions with no data
language	character. 2 letter language code for output labels

### Value

data.frame in long format (wide not yet supported). Contains attributes for the URL and parameters used.

### Examples

```

## Not run:

# Get data for Cropland (6620) Area (5110) in Antigua and Barbuda (8) in 2017
df = read_fao(area_codes = "8", element_codes = "5110", item_codes = "6620",
  year_codes = "2017")
# Load cropland area for a range of year
df = read_fao(area_codes = "106", element_codes = "5110", item_codes = "6620",
  year_codes = 2010:2020)

```

```
# Find which country codes are available
metadata_area <- read_dimension_metadata("RL", "area")
# Find which items are available
metadata_item <- read_dimension_metadata("RL", "item")
# Find which elements are available
metadata_element <- read_dimension_metadata("RL", "element")

## End(Not run)
```

---

scaleUnit

*A function to standardize the unit*

---

## Description

The function standardize the data to the desirable unit when the multiplier vector is supplied. For example per 1000 people is scaled to per person by supplying a multiplier of 1000.

## Usage

```
scaleUnit(df, multiplier)
```

## Arguments

df	The data frame containing the data to be scale
multiplier	The named vector with the multiplier to be scaled. The name is mandatory in order for the function to identify the variable in the data frame. A data.frame can also be supplied with the first column being the name and the second being the numeric multiplier.

## Examples

```
## Create the data frame
test.df = data.frame(FAOST_CODE = 1:5, Year = 1995:1999,
  var1 = 1:5, var2 = 5:1)

## Create the named vector for scaling
multiplier = c(1, 10)
names(multiplier) = c("var1", "var2")

## Scale the data
scaleUnit(test.df, multiplier = multiplier)
```

---

search_dataset	<i>Search FAOSTAT dataset_codes, datasets, elements, indicators, and items</i>
----------------	--

---

### Description

Get full list of datasets from the FAOSTAT database with the Code, dataset name and updates.

### Usage

```
search_dataset(dataset_code, dataset_label, latest = TRUE, reset_cache = FALSE)
```

```
FAOsearch(dataset_code, dataset_label, latest = TRUE, reset_cache = FALSE)
```

### Arguments

dataset_code	character. Code of desired datasets, listed as 'code' in output.
dataset_label	character. Name of the datasets, listed as 'label' in the output data frame. Can take regular expressions.
latest	logical. Sort list by latest updates
reset_cache	logical. By default, data is saved after a first run and reused. Setting this to true causes the function to pull data from FAO again

### Value

A data.frame with the columns: code, label, date\_update, note\_update, release\_current, state\_current, year\_current, release\_next, state\_next, year\_next

### Examples

```
## Not run:  
# Find information about all datasets  
fao_metadata <- search_dataset()  
# Find information about forestry datasets  
search_dataset(dataset_code="FO")  
# Find information about datasets whose titles contain the word "Flows"  
search_dataset(dataset_label="Flows")  
  
## End(Not run)
```

---

shConstruct	<i>Construct share variable</i>
-------------	---------------------------------

---

### Description

A function for constructing the share of a variable of an aggregated variable.

### Usage

```
shConstruct(data, totVar, shareVar, newVarName = NA)
```

### Arguments

data	The data frame containing both the share variable and the aggregated variable
totVar	The aggregated variable.
shareVar	The subset of the aggregated variable which to be divided by.
newVarName	The name assigned to the new variable, if missing then <i>.SC/.SH/.GR</i> will be appended depending on the type of construction

### Details

The share of a variable can be share of the World (if additional variable were not supplied) or share of another variable (per Capita if population was supplied).

### Value

A data frame with the new constructed variable

### Examples

```
## Total variables provided, scale by totVar
test.df = data.frame(FAOST_CODE = 1, Year = 1990:1994, a = 1:5, b = 1:5)
shConstruct(data = test.df, totVar = "a", shareVar = "b")

## Total variables not provided, scale by world aggregate.
test.df2 = data.frame(FAOST_CODE = rep(c(1, 5000), each = 5),
  Year = rep(1990:1994, 2),
  a = rep(1:5, 2), b = rep(1:5, 2))
shConstruct(data = test.df2, totVar = NA, shareVar = "b")
```



---

translateUnit	<i>Function to translate multipliers</i>
---------------	--

---

**Description**

This function translates number to character name or vice versa

**Usage**

```
translateUnit(vec)
```

**Arguments**

vec                    The vector containing name or number to be translated

**Examples**

```
## Create numeric vector
myUnit = c(1000, 1e6, 1000, 1e9, 1e9, 1e12)

## Translate numeric to character
myUnit2 = translateUnit(myUnit)
myUnit2

## Now translate back
translateUnit(myUnit2)
```

---

translate_countrycodes	<i>A function to translate between different country coding systems</i>
------------------------	---

---

**Description**

The function translate any country code scheme to another if both are in the are among the types present in the FAO API. If you require other codes or conversion of country names to codes, consider the countrycodes package.

**Usage**

```
translate_countrycodes(
  data,
  from = c("FAO", "M49", "IS02", "IS03"),
  to = c("M49", "FAO", "IS02", "IS03", "name"),
  oldCode,
  reset_cache = FALSE
```

```
)  
  
translateCountryCode(  
  data,  
  from = c("FAO", "M49", "IS02", "IS03"),  
  to = c("M49", "FAO", "IS02", "IS03", "name"),  
  oldCode,  
  reset_cache = FALSE  
)
```

**Arguments**

<code>data</code>	The data frame
<code>from</code>	The name of the old coding system
<code>to</code>	The name of the new coding system
<code>oldCode</code>	The column name of the old country coding scheme
<code>reset_cache</code>	logical. Whether to pull data from FAOSTAT directly instead of caching

# Index

- \* **datasets**
  - FAOcountryProfile, 11
  - FAOmetaTable, 11
  - FAOregionProfile, 11
- \* **package**
  - FAOSTAT-package, 2
- Aggregation, 3
- change\_case, 4
- chConstruct, 5
- check\_country\_overlap, 6
- chgr, 6
- CHMT, 7
- constructSYB, 8
  
- download\_faostat\_bulk, 9
  
- ebind, 11
  
- FAOcountryProfile, 11
- FAOmetaTable, 11
- FAOregionProfile, 11
- FAOsearch (search\_dataset), 23
- FAOSTAT (FAOSTAT-package), 2
- FAOSTAT-package, 2
- fillCountryCode, 12
  
- geogr, 12
- get\_faostat\_bulk
  - (download\_faostat\_bulk), 9
- getFAO, 14, 15
- getFAO (read\_fao), 20
- getWDI, 13, 15
- getWDImetaData, 14
- getWDItoSYB, 14, 15
- grConstruct, 16
  
- indConstruct, 17
  
- lsgr, 17
  
- make.names, 4, 5
- mergeSYB, 18
  
- overlap, 19
  
- printLab, 19
  
- read\_bulk\_metadata
  - (download\_faostat\_bulk), 9
- read\_dataset\_dimension, 20
- read\_dimension\_metadata
  - (read\_dataset\_dimension), 20
- read\_fao, 20
- read\_faostat\_bulk
  - (download\_faostat\_bulk), 9
  
- scaleUnit, 22
- search\_dataset, 20, 23
- shConstruct, 24
  
- translate\_countrycodes, 25
- translateCountryCode
  - (translate\_countrycodes), 25
- translateUnit, 25